DISCRETE OPTIMIZATION ALGORITHMS FOR MARKER-ASSISTED PLANT BREEDING

HERMAN DE BEUKELAER

Promotors:
Prof. Dr. Veerle Fack
Dr. Geert De Meyer

Thesis submitted to obtain the degree of
Doctor of Science: Computer Science
2016–2017

Department of Applied Mathematics, Computer Science and Statistics
Faculty of Sciences, Ghent University

Research funded by the Research Foundation - Flanders (FWO)

The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, the Hercules Foundation and the Flemish Government—department EWI.

Herman De Beukelaer
Discrete Optimization Algorithms for Marker-Assisted Plant Breeding

PREFACE

Before diving into optimization theory and plant breeding, I would like to take the opportunity to thank some people without whom I would not have been able to complete this doctoral thesis.

As a master student, Veerle offered me the opportunity to write a thesis on optimization algorithms for plant breeding. When accepting this offer, I had no idea that it would be the start of a fruitful six-year collaboration with the biometrics team at Bayer CropScience. At the time, this team consisted of two people only, Geert and Katrien, who introduced me to the fascinating world of plant breeding and genetics, and guided me through my master thesis and towards my PhD research. Veerle, Geert, and Katrien, thanks for being there for me from the start.

Guy, you joined us a few months after I threw myself with full enthusiasm on the further development of Core Hunter. It still seems like an amazing coincidence that, just then, you, one of the original authors of Core Hunter, got a job at Bayer in Zwijnaarde. I learned a lot from you, for which I am very grateful. Although you recently moved to the other side of the world, our collaboration continues. What started as a master thesis grew to become a successful research collaboration, and Core Hunter will soon turn into one of my pet projects.

As the biometrics team at Bayer grew, and became the biometrics and breeding research team, more people joined the club. Yvonne, thanks to you I was able to work on a fascinating research project about long-term genomic selection, and together we wrote a great paper which I would not even have understood before, without your help. A huge thanks is reserved for Geert as well here, for taking over when Yvonne left Bayer and sacrificing so much of his free time to this project. Together we have come a long way, with great results. I cannot thank you enough for this.

Veerle, Geert, Katrien, Guy, and Yvonne, your guidance and support have been amazing! I really enjoyed working with you and highly appreciate everything you have done for me.

I also want to thank all members of the examination committee: Jean-Luc Jannink, Guy Davenport, Yvan Saeys, Pieter Audenaert, Geert De Meyer, Veerle Fack, and Peter Dawyndt. I hope you enjoyed reading my work and am very grateful for the many sugges-

tions for improving my dissertation, and the interesting discussion we had during my defense.

To my colleagues, I would like to say that I really enjoyed working in the department. A big thanks to all of you for the fun we had during lunch, coffee breaks, and the many after-work events. Bart, Jan, and Felix, I loved teaching our students about algorithms and data structures, together with you. In particular, I am very grateful to Felix for relieving me from a great deal of my teaching assignment, when I was short on time while writing my thesis and preparing for the defense.

Needless to say—though ever so important—I am extremely grateful to my parents too, who have always supported me, from child to student and beyond. It's far to easy to take the chances you gave me for granted. I hope you are proud of what I have accomplished, but you should be equally proud of yourself for leading the way.

Ruth, my lovely wife and soon-to-be-mother of our firstborn child, you're the best thing that ever happened to me. Thanks for taking care of me and accepting me the way I am. Whenever I lose myself in my perfectionism, you're always there to put me back on track. Together we can achieve anything, and there is nothing that I desire more than to spend the rest of my life with you.

Herman De Beukelaer
May 2017

PUBLICATIONS

De Beukelaer, Herman, Petr Smỳkal, Guy F Davenport and Veerle Fack (2012). „Core Hunter II: fast core subset selection based on multiple genetic diversity measures using Mixed Replica search". In: *BMC bioinformatics* 13.1, p. 312.

De Beukelaer, Herman, Geert De Meyer and Veerle Fack (2015a). „Heuristic exploitation of genetic structure in marker-assisted gene pyramiding problems". In: *BMC genetics* 16.1, p. 1.

De Beukelaer, Herman, Guy F Davenport, Geert De Meyer and Veerle Fack (2015b). „JAMES: A modern object-oriented Java framework for discrete optimization using local search metaheuristics". In: *4th International symposium and 26th National conference on Operational Research*. Hellenic Operational Research Society, pp. 134–138.

De Beukelaer, Herman, Guy F Davenport and Veerle Fack (2017a). „Core Hunter 3: flexible core subset selection". In: *Submitted to BMC bioinformatics*.

De Beukelaer, Herman, Guy F Davenport, Geert De Meyer and Veerle Fack (2017b). „JAMES: An object-oriented Java framework for discrete optimization using local search metaheuristics". In: *Software: Practice and Experience* 47.6, pp. 921–938. DOI: 10.1002/spe.2459.

De Beukelaer, Herman, Yvonne Badke, Veerle Fack and Geert De Meyer (2017c). „Moving Beyond Managing Realized Genomic Relationship in Long-Term Genomic Selection". In: *Genetics*. DOI: 10.1534/genetics.116.194449.

CONTENTS

ACRONYMS

A-NE Average accession-to-nearest-entry distance

API Application programming interface

CH1 Core Hunter 1

CH2 Core Hunter 2

CH3 Core Hunter 3

DAG Directed acyclic graph

DH Doubled haploid

DMIN Minimum distance

DNA Deoxyribonucleic acid

E-NE Average entry-to-nearest-entry distance

GA Genetic algorithm

GBS Genotyping-by-sequencing

GDOpt Genetic distance optimization

GEBV Genomic estimated breeding value

GM Genetic modification

GMO Genetically modified organism

GOCS Genomic optimal contributions selection

GP Genomic prediction

GS Genomic selection

GWAS Genome-wide association

HE Expected heterozygosity

IBD Identity-by-descent

IBS Identity-by-state

IQR Inter-quartile range

LPA Linkage phase ambiguity

MixRep Mixed replica search

OC Optimal contributions

OCS Optimal contributions selection

QTL Quantitative trait locus/loci

REMC Replica exchange Monte Carlo search

SNP Single nucleotide polymorphism

SSR Simple sequence repeat

TP Training population

TSP Travelling salesman problem

TSPLIB Travelling salesman problem library

WGS Weighted genomic selection

Part I

INTRODUCTION

This introductory part provides a general background of plant breeding and optimization, that will aid the reader to understand the breeding problems addressed in subsequent chapters, and the optimization techniques used to solve these problems.

1

INTRODUCTION TO PLANT BREEDING

Some insight into traditional and modern plant breeding is needed to be able to understand the problems addressed in chapters 4 to 6. We introduce most relevant concepts here, loosely based on selected chapters from *Principles of plant genetics and breeding* by Acquaah (2012), and refer the reader to this excellent book for more information about plant breeding, genetics and related technologies.

1.1 HISTORY OF PLANT BREEDING

More than ten thousand years ago our ancestors started to herd animals and cultivate plants, such as vegetables and field crops, as they left behind their lives as hunters and gatherers. These ancient farmers did not only invent agriculture but were also the first plant breeders, as evidence was found that they stored seed from good looking plants and used it for planting in the next season. In this way, heritable favourable traits were propagated and accumulated over generations leading to better variants in each season. Over time, cultivated crops became more and more adapted to the environment in which they were grown, for example flowering and ripening at appropriate times (e. g. before winter). Such local, highly adapted varieties obtained through decades of purely selection are commonly referred to as *landraces* and provide a useful source of diverse material to be exploited in various breeding programs.

The goal of plant breeders is to continuously improve certain traits of cultivated crops, vegetables, trees, flowers, or any plant of their interest. Important traits include resistance to diseases and abiotic stress—such as wind, extreme temperatures, drought or flood—and many quantitative traits—such as yield, or certain consumer preferences like taste, high nutritional quality, shape or colour. The most fundamental technique used for this purpose has always been and still is artificial selection. Of course, selection requires variability to discriminate among. Early plant breeders depended on the diversity of wild species and the naturally occurring variability on the field as a result of *cross-pollination*. As the wind blows and insects like bees fly from one flower to another, they carry pollen and randomly fertilize plants leading to natural diversity among the offspring. To really control the process, one thus needs to make

artificial crossings, which turned out to be more difficult to master than the selection step. There is some archeological evidence that the Babylonians and Assyrians manually pollinated date palms by transferring pollen from a male plant to fertilize flowers on a female plant, but it took several more millennia to really unravel the sexuality and heredity of plants and to go beyond the rudimentary plant manipulations as occasionally done by these early civilizations.

In contrast to date palms, and some other plants like pistachio trees and kiwi vines, most plant species are bisexual, having flowers that contain both the male (stamens) and female parts (pistils) or sometimes separate male and female flowers on the same plant. Such plants can usually be self-pollinated, as is the case for most field crops and vegetables. Some species however, like most apple and several other kinds of fruit trees, still require cross-pollination for successful fertilization, even though apple flowers are bisexual.

In the late 17th century Rudolph Camerarius, a German botanist from the University of Tubingen, described the male and female reproductive parts of plants and their function in fertilization, in his work *De sexu plantarum epistola* (Camerarius, 1694). He conducted experiments with several species, including mulberry and maize, to validate the general belief that plants had some form of sex and that pollen were the male fertilizer. When growing unisexual, female mulberry trees outside the vicinity of male trees, no seed was produced on the female plants. Likewise, by taking away the male flowers of maize plants, no seed was formed as fertilization was prevented. In the following century, Joseph Gottlieb Kölreuter, also a German botanist, experimented with artificial fertilization. He performed systematic crossings of tobacco plants and discovered that, on average, the hybrid offspring resembles both parents equally. Kölreuter also recognized the role of wind and insects in natural pollination.

In the 19th century Louis de Vilmorin investigated how to develop new plant varieties with specific characteristics through repeated crossing (hybridization) and selection. He developed a basic theory of heredity, recognizing that selected traits are passed on and accumulated through generations. As such, his work laid the foundation for the modern breeding industry. The great grandfather of Louis de Vilmorin was chief botanist and seed supplier for King Louis XV and until today, the Vilmorin company is a major French seed producer, which was family-owned for about two centuries.

Around the same time Charles Darwin, an English naturalist and geologist with one of the most well-known names in the history of natural sciences, developed his theory of evolution, according to which all species of life have evolved from a common ancestor. He published his work in the famous book *On the origin of species* (Darwin, 1859). Evolution is driven by natural selection due to survival of the fittest individuals, that happen to be best adapted to their environment. As mentioned before, selection requires variability, and the main source of variation for natural evolution are rare genetic mutations which by chance turn out to be favourable. Therefore, evolution happens extremely slowly, requiring thousands or millions of years to diverge new species from their common ancestor.

Natural selection also acts within populations of existing species, where the fittest individuals have the best chance for survival and reproduction, due to which they produce the largest offspring and keep the population fitness intact. In a way, plant breeding can be seen as a controlled high-speed emulation of evolution. By performing well-chosen crossings followed by artificial selection, breeders are able to nudge nature to their advantage.

Figure 1.1 shows a schematic overview of a traditional so-called *recurrent selection* plant breeding scheme. First, the breeder gathers diverse material used to initiate the scheme, for example two or more existing high-quality varieties, possibly with complementary favourable traits. This material is then used for repeated crossing, evaluation and selection. At the end of each generation, the selection leads to a new product that the breeder can sell to his customers. In addition, the selection is advanced to the next generation for further improvement. In contrast to landraces, a variety produced through deliberate plant breeding is called a *cultivar*.

Traditional plant breeding is mostly based on intuition and experience of the breeder who repeatedly selects plants with favourable observable characteristics. Therefore, plant breeding is often said to be an art, requiring a keen eye and well-developed gut feeling, even more so because some traits are difficult to observe and the performance of plants for example also depends on the environment in which they were grown. Over the last two centuries however, scientists started to unravel the genetics behind the expressed traits and the mechanisms underlying heredity, i. e. how traits are passed on from parents to their offspring. Many technological advances followed, especially during the last few decades, that now make it possible to directly improve the genetic composition of plants. As such, plant breeding has been and is still further being transformed from an art into a science—the science of molecular breeding.

1.2 MOLECULAR BREEDING

1.2.1 *Basics of genetics*

Mendel started his experiments on heredity with mice. The abbey's bishop however did not like that one of his monks was studying animal sex, so Mendel continued his work with plants.

Another famous natural scientist from the 19th century is Gregor Mendel, who is often considered to be the father of modern genetics. Mendel was born in Hynčice—at that time part of the Austrian Empire, now of the Czech Republic—and joined the Augustinian Abbey of St Thomas in Brno. Born in a family of farmers, and fascinated by science, becoming a friar allowed Mendel to receive a high-level education without having to pay for it himself. In the

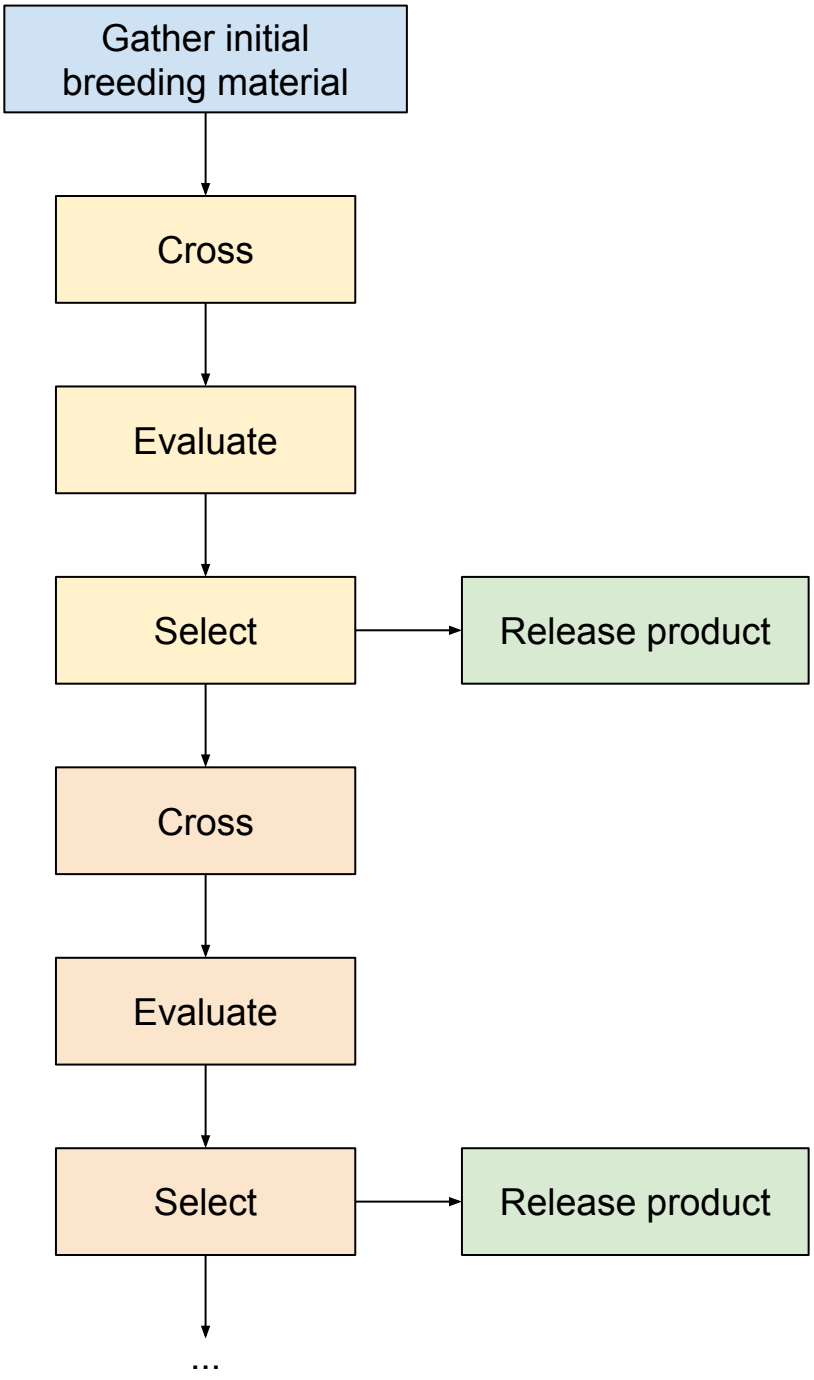


Figure 1.1: Schematic overview of a traditional so-called *recurrent selection* plant breeding scheme. Two generations of crossing, evaluation and selection are displayed. After each selection step, a new product is released, and the chosen plants are advanced to the next generation for further improvement.

monastery, he performed crossings with pea plants and carefully recorded the number of variants among the offspring, looking at several characteristics that seemed to inherit independently, such as plant height, seed colour and shape, and flower colour. Based on his observations he concluded that there are invisible units of heredity, which he called "factors" and are now called *genes*, that occur in alternate "forms"—now called *alleles*—and control the observed traits in predictable ways. Moreover, these genes follow what are currently known as the three laws of Mendelian inheritance: segregation, independent assortment and dominance.

Mendel described that an individual contains two alleles of each gene, one inherited from each parent, a property which is now referred to as *diploidy*. Alleles segregate when an organism produces reproductive cells—sperm and eggs—meaning that these so-called *gametes* contain only half of the genetic information from the parent, i. e. one allele for each gene. Fusion of two *haploid* gametes of the opposite sex produces a new *diploid* individual. Chance determines which combination of alleles is formed in the gametes and passed on to the offspring. Although Mendel concluded that different traits are independently inherited and as such, that each possible allele recombination occurs with equal probability, this is not always the case, due to genetic linkage, which is explained below. Finally, Mendel's law of dominance states that, while an individual can contain two different alleles of the same gene, they are not necessarily both expressed as observable characteristics. One allele may be dominant and as such mask the presence of the other, so-called recessive allele. Still, the recessive allele can be passed on to and expressed in future offspring.

Cystic fibrosis is an example of a human genetic disease controlled by a recessive mutation in a single gene. A healthy person may carry one mutated allele as the disease only develops when both copies are mutated. If two healthy parents happen to be both carrier, they have a chance of one out of four to have a child with cystic fibrosis.

It was later discovered that the cells of each known living organism contain so-called DNA (deoxyribonucleic acid) that carries the genes described by Mendel. In *eukaryotic* cells, such as those of humans, animals and plants, most DNA is found in the *nucleus* (figure 1.2). This nuclear DNA is diploid and structured into multiple *chromosomes* containing the genes that follow the laws of Mendelian inheritance. In the mid-20th century Watson and Crick (1953) showed that DNA molecules have a double helical structure. Both strands of the double helix consist of a long sequence of *nucleotides*, four of which occur in DNA: cytosine (C), guanine (G), adenine (A) and thymine (T). The order in which the nucleotides appear determines the genetic architecture of an organism, and certain sequences of nucleotides spread across the DNA encode for genes that may be expressed as various observable characteristics. Both strands actually each contain all information, as A always binds with T, and C with G. This redundancy may seem a bit weird

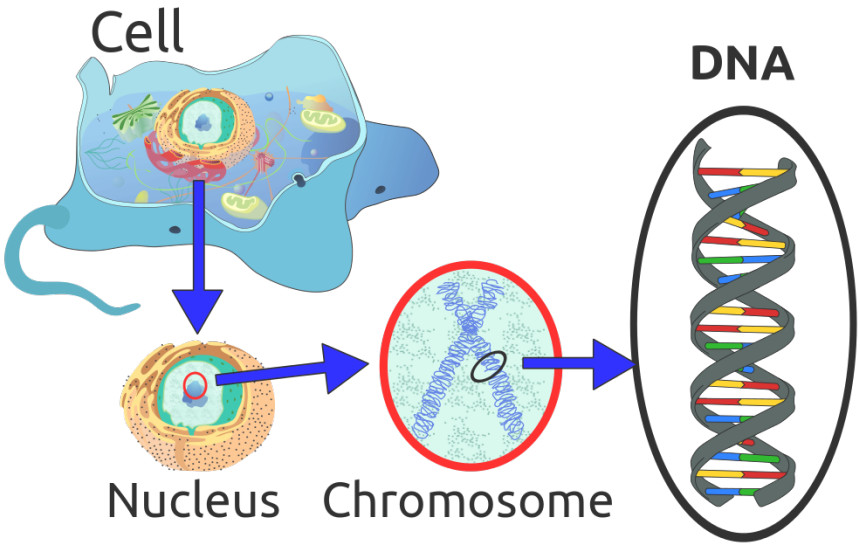


Figure 1.2: A cell of an eukaryotic organism has a well-delimited core, called the nucleus, which contains most of the DNA. This DNA is separated into a set of chromosomes and has a double helical structure composed of two complementary sequences of nucleotides: adenine (A; red), thymine (T; yellow), guanine (G; blue) and cytosine (C; green). The small piece of DNA shown in detail is characterized by one of the two complementary strings AGGATGCTACGATCTGTG or TCCTACGATGCTAGACAC (read from top to bottom).

but it is just one of nature's clever tricks used to duplicate DNA by separating the strands and regrowing the other half.

Confirming Mendel's theory of diploidy, chromosomes always come in pairs, each consisting of two so-called *homologous* chromosomes—one inherited from the father and one from the mother. For example, humans have 23 chromosome pairs (figure 1.3). An individual can thus either contain twice the same allele (*homozygous*) or two different alleles (*heterozygous*) of a particular gene. When producing sperm and egg cells the homologous chromosomes recombine—a process called *crossover*—which causes segregation of genes as observed by Mendel (figure 1.4). Although Mendel suggested that different traits segregate independently, this is only true when the corresponding genes lie on different chromosomes or at a certain distance on the same chromosome, which was the case for all the traits he studied. Genes located close to each other on the same chromosome are *linked* and those with a higher *genetic linkage* are less likely to recombine, because the area in which a crossover is needed becomes smaller as the genes lie closer together. The distance between genes can be measured as a physical distance,

expressed in number of base pairs, or as a genetic distance, expressed in *centimorgans* (cM) named after geneticist Thomas Hunt Morgan. The latter corresponds to the recombination rate, which is approximately d% for genes at a small distance of d cM and goes to 50% as d goes to infinity. By performing experimental crosses and counting the number of recombinations, scientist are able to create *genetic maps* that describe the positions (*loci*) of and distances between multiple genes of a certain organism.

The complete set of an organism's genetic material is called its *genome*, including the parts of the DNA that do not code for genes and any non-nuclear DNA. The term *genotype* is used to refer specifically to the collection of genes, which determine the characteristics of an individual—the set of these observable traits is called the *phenotype*. A genotype encodes for a certain phenotype and while traditional breeding is based on phenotypic selection, modern breeding aims to focus directly on improving the genotype, containing the actual heritable genes responsible for the observed variability.

Through genome-wide association studies (GWAS) scientists are able to identify the effect of certain genes and their alleles on specific traits (Cantor et al., 2010). It turned out that there are simple traits which are controlled by one or a few genes only, for example including many disease resistances, while quantitative traits, such as size or yield, are generally affected by many genes (up to multiple thousands) that are spread across the genome and each have a relatively small additive effect. The latter is known as the *infinitesimal model* of quantitative genetics first described by the famous statistician Fisher (1919). To improve the quality of a plant in terms of such complex quantitative trait one thus needs to accumulate beneficial alleles of many so-called *quantitative trait loci* (QTL) to maximize their total effect.

1.2.2 *Molecular markers*

Many methods have been developed to extract DNA from organisms such as plants, animals or humans—a process called *DNA sequencing*. Since each cell of an individual contains a copy of its full DNA, sequencing generally requires only a small amount of material. In case of plants it is usually sufficient to cut off a leaf while for animals a blood sample is taken. However, current technology does not allow to sequence an organism's entire genome at once. Instead, the DNA is read in parts that have to be stitched together, which is a tedious and costly process. Genotyping-by-sequencing (GBS; Poland and Rife 2012) recently became feasible for some applica-

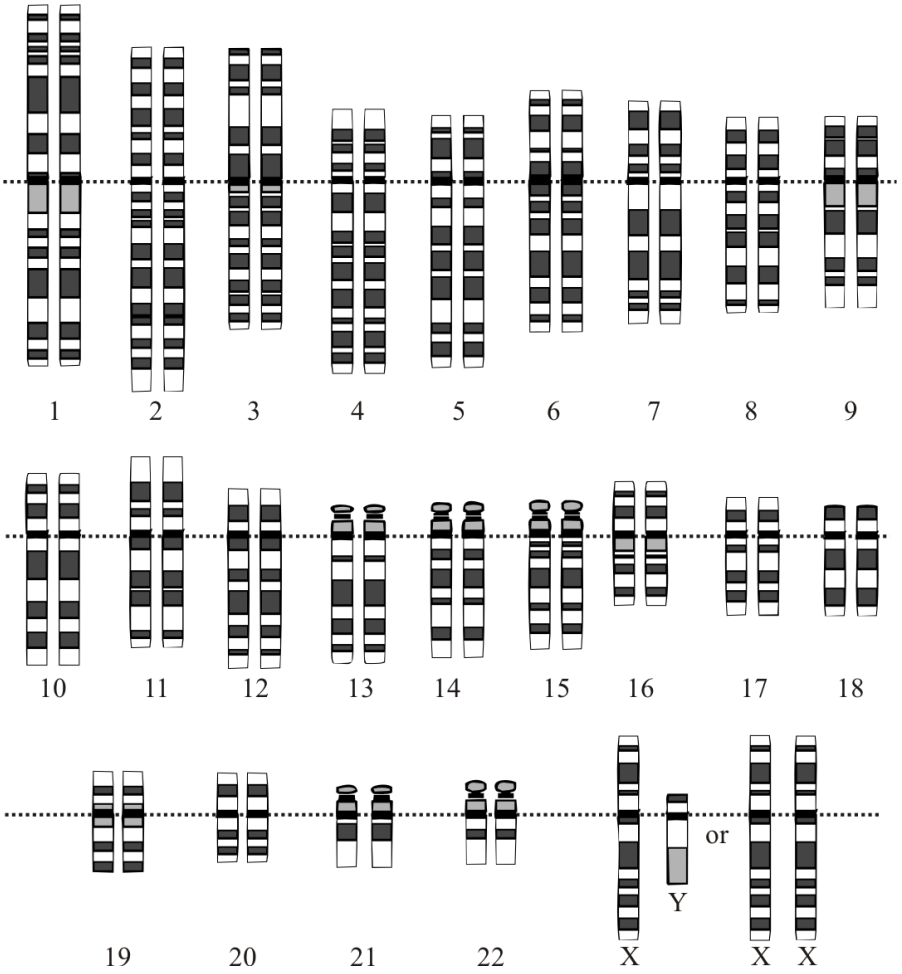


Figure 1.3: Human DNA is composed of 23 chromosome pairs, each consisting of two homologous chromosomes, one inherited from both parents. The last pair differs between men, who have one X and one Y chromosome, and women, who have two X chromosomes. When producing sperm and egg cells the homologous chromosomes recombine which results in the segregation of genes as described by Mendel. As an exception, only the tips of the X and Y chromosome are able to recombine, which means that the Y chromosome is passed on almost identically from a father to all of his suns, cheating a bit on the laws of Mendel.

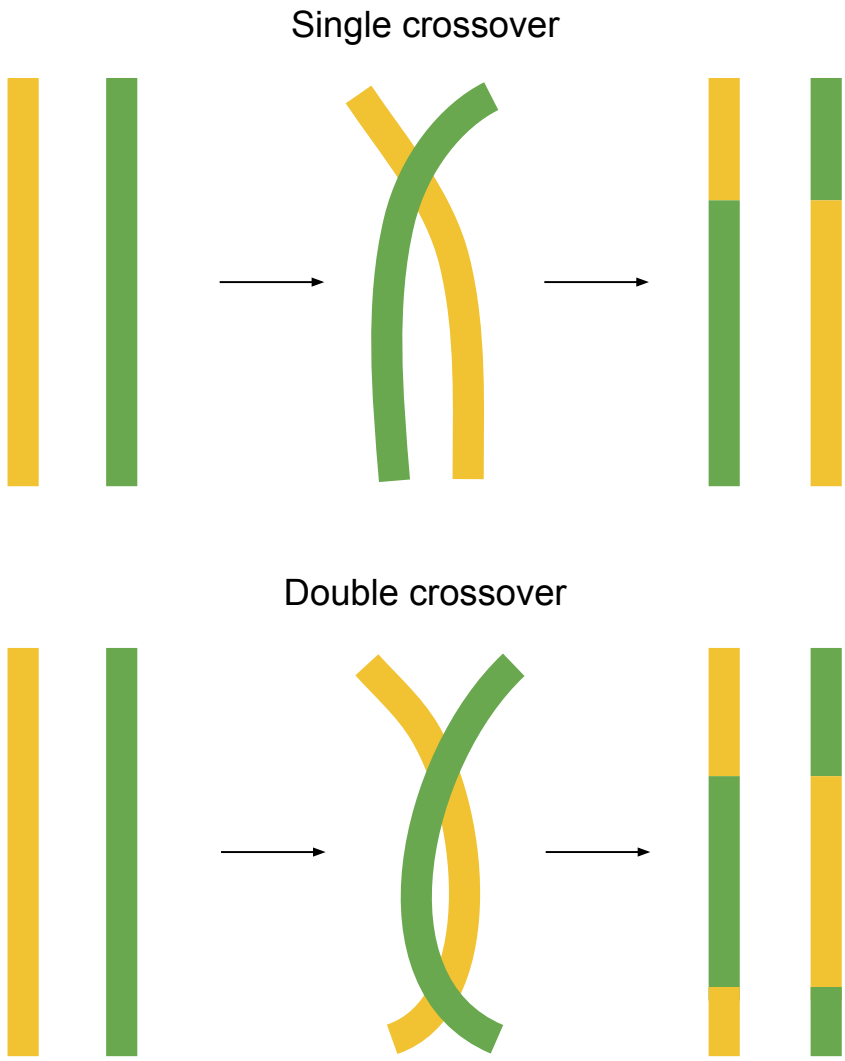


Figure 1.4: When an organism produces gametes—sperm and egg cells—homologous chromosomes may recombine at one or more locations. This mechanism is the source of diversity within species and explains the segregation of traits that was observed by Mendel. Here, an example of a single and double crossover is shown, resulting in gametes with different recombinants.

tions due to steadily decreasing sequencing costs, but is generally still quite expensive. Most commonly, including for GBS, *molecular* or *genetic markers* are used to characterize a genome, because these are easy to work with and can also be extracted without full DNA sequencing through various methods (Semagn et al., 2006).

Genetic markers are easily observable landmarks in the genome—short sequences of DNA that vary among individuals, similar to genes, but without encoding for phenotypic traits. Following the terminology of genes, the variants of a marker are referred to as its alleles, and an individual can be homozygous or heterozygous at a certain marker locus. Sometimes particular genes are tracked by tagging them with closely linked markers, while other applications use dense markers spread across the entire genome.

Many different marker systems have been developed over the years, with varying properties. For several of these marker types, genetic maps were constructed for many plant (and animal) species. Two examples of frequently used markers are *microsatellites* (or simple sequence repeats; SSRs) and *single nucleotide polymorphisms* (SNPs). Microsatellites are short repetitive DNA sequences, which generally have several possible alleles, corresponding to a different number of repeats. For example, one individual may contain the sequence GAC while another has GACGAC, or GACGACGAC, etc. This specific SSR would be referred to as $(GAC)_n$ where n differs among alleles. On the other hand, SNPs correspond to differences in a single nucleotide. Although, in theory, SNPs can have up to four alleles (A, T, G, C), they are actually mostly bi-allelic.

Regardless of the employed marker system, an individual (table 1.1) or group (table 1.2) can be characterized by reporting the observed allele frequencies for a collection of markers. However, bi-allelic marker data can be represented more compactly, for example as a matrix with allele scores 0, 1 and 2 (table 1.3). This is possible because for a marker with two alleles, say A and a, there are only three combinations: AA (0), Aa (1) and aa (2). As such, the value in the marker matrix indicates the number of copies of a certain reference allele (here a).

For some applications, knowledge of the *linkage phase*—which allele resides on which homologous chromosome?—may be required to discriminate between different heterozygous genotypes with the same allele frequencies. For example, the second genotype in table 1.3, with allele scores $[2, 1, 1]$, could have linkage phase

$$\text{(a)} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{or} \quad \text{(b)} \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

	M1-1	M1-2	M1-3	M2-1	M2-2	M3-1	M3-2	M3-3
G1	0.00	1.00	0.00	0.50	0.50	0.00	0.50	0.50
G2	1.00	0.00	0.00	0.50	0.50	0.00	0.50	0.50
G3	0.50	0.00	0.50	0.00	1.00	0.50	0.50	0.00
G4	0.00	0.00	1.00	1.00	0.00	0.50	0.00	0.50
G5	0.00	0.50	0.50	0.50	0.50	0.50	0.50	0.00

Table 1.1: Example of allele frequency data for five individuals (rows) that were genotyped using three multi-allelic genetic markers (columns) having three, two, and three alleles, respectively. Because we are looking at a diploid organism, only frequencies 0.0, 0.5, and 1.0 were observed. A frequency of 1.0 means that an individual is homozygous for the respective marker, having twice the same allele. On the other hand, if an individual is heterozygous at a specific marker, the two respective alleles are observed with a frequency of 0.5. The values for each marker sum to 1.0 in each individual.

In the first case, the 1-alleles at the second and third locus reside on the same homologous chromosome, while they are separated in case of the second linkage phase. Even though the linkage phase does not affect the expressed phenotype, it determines the distribution of produced gametes and therefore the probability to observe a certain genotype in the offspring of a crossing. If the second and third marker in the example above happen to be genetically linked, a genotype with linkage phase (a) will more likely produce gametes $[1, 0, 0]$ and $[1, 1, 1]$, and fewer gametes $[1, 0, 1]$ or $[1, 1, 0]$, as compared to a genotype with linkage phase (b). Unfortunately, the linkage phase is not easily observed with current genotyping technologies, but it can be inferred or estimated by looking at additional information such as the genotypes of an individual's ancestors in its *pedigree* (the "family tree").

1.2.3 *Marker-assisted selection*

Breeders can use marker data to make better decisions as compared to traditional breeding based solely on phenotypic selection. For example, the availability of marker data makes it possible to compose genetically diverse breeding material, to perform better crossings, or to select sooner and with higher accuracy among the produced offspring in each generation. Over the last decade several marker-based plant breeding strategies have been established and are increasingly used to develop better products. At present, geno-

	M1-1	M1-2	M1-3	M2-1	M2-2	M3-1	M3-2	M3-3
G1	0.00	0.90	0.10	0.60	0.40	0.00	0.25	0.75
G2	1.00	0.00	0.00	0.50	0.50	0.00	0.50	0.50
G3	0.60	0.00	0.40	0.05	0.95	0.30	0.70	0.00
G4	0.20	0.05	0.75	1.00	0.00	0.20	0.00	0.80
G5	0.33	0.33	0.33	0.15	0.85	0.55	0.45	0.00

Table 1.2: Example of allele frequency data for five groups of $n = 30$ individuals (rows) that were genotyped using three multi-allelic genetic markers (columns) having three, two, and three alleles, respectively. In a group of n diploid individuals, $2n$ alleles are observed for each marker, which means that all multiples of $\frac{1}{2n}$ are possible frequencies. Again, the values for each marker sum to 1.0 in each individual.

typing technologies are no longer limiting and the major challenge is to optimally use genetic markers in practical breeding schemes.

One very promising marker-assisted selection technique used for complex quantitative traits is to predict breeding values from genetic marker data. Recording phenotypic traits is a very time consuming and expensive task, further complicated by the fact that observable characteristics are influenced by the environment. Also, some traits are difficult to observe, and often adult plants are needed. Ideally, a breeder wants to select plants with a high *genetic value*—which will be inherited by the offspring produced in subsequent crossings—as soon as possible in the breeding cycle.

Using datasets for which both genetic marker data and phenotypes are available, statistical regression models can be built that predict the genetic value of an individual from its marker data. Such models are called *genomic prediction* models and selecting based on the predicted values is referred to as *genomic selection* (GS)—a major trend in modern marker-assisted breeding for complex quantitative traits (Heffner et al., 2009). Because there are so many, it is difficult to identify all QTL affecting a complex trait of interest. Genomic selection offers a practical solution by using a large number (up to tens or hundreds of thousands) of genome-wide markers in the hope that most QTL effects are picked up in the prediction model through a linked marker in its vicinity. A major practical advantage of genomic selection is that it allows to perform more selection cycles per time unit. Once a prediction model has been obtained, phenotypic evaluation does not need to be completed before selection takes place. Yet, if necessary, evaluation can be performed in

Allele frequencies for bi-allelic markers

	M1-1	**M1-2**	**M2-1**	**M2-2**	**M3-1**	**M3-2**
G1	0.50	0.50	0.50	0.50	0.50	0.50
G2	1.00	0.00	0.50	0.50	0.50	0.50
G3	0.50	0.50	0.00	1.00	1.00	0.00
G4	0.00	1.00	1.00	0.00	0.00	1.00
G5	0.50	0.50	0.50	0.50	1.00	0.00

Allele scores

	M1	**M2**	**M3**
G1	1	1	1
G2	2	1	1
G3	1	0	2
G4	0	2	0
G5	1	1	2

Table 1.3: Example of allele frequency data (top) for five individuals (rows) that were genotyped using three bi-allelic markers (columns). Now there are only two columns per marker, with values 0.0, 0.5, and 1.0, again summing to 1.0 for each marker. Therefore, one column can obviously be inferred from the other, and it is possible to more compactly represent the data as a matrix (bottom) with one row per individual and one column per marker. Values in this matrix are allele scores 0, 1, and 2, and indicate the number of observed copies of an arbitrary reference allele. In this example, the first allele of each marker was taken as the reference allele.

parallel with selection to update the prediction model in order to improve its accuracy in the next cycle (figure 1.5).

For simple traits, controlled by a small number of genes, we can go much further than prediction of value from marker data. Here, the genetic profile of the breeding target can be fully defined at the few involved loci. It is therefore possible to predesign a detailed crossing scheme that obtains this target genotype from the available parents in an efficient way, for example minimizing the associated cost and number of generations. Such crossing scheme tells the breeder precisely which plants to cross in each generation and which genetic profiles, in terms of the involved genes, to select from the offspring. Due to the massive number of possible crossing schemes, that quickly increases when considering more genes, such an approach is only feasible for simple traits controlled by up to a dozen of genes.

Of course many other marker-assisted selection techniques have been and are being developed besides those that we addressed. For example, marker-assisted backcrossing is used to transfer selected genes, e. g. from a wild relative, into an elite background. By tracking a large number of background markers, breeders can assure that only the desired genes are transferred with as little surrounding wild DNA as possible, to retain the high quality of the elite product. For more information about modern marker-assisted breeding techniques we refer to Tester and Langridge (2010).

Although plant breeding used to be mainly an art, science is now increasingly taking the guesswork out. In this respect, marker-assisted selection is one of the major technologies with huge potential to lead to further successes in coping with a changing environment and a vastly growing world population.

1.2.4 *Other molecular breeding techniques*

In contrast to marker-assisted selection which is used to improve conventional breeding driven by repeated crossing and selection, other methods have been developed that directly modify the genetic composition of plants (or animals)—a practice referred to as *genetic engineering* or *genetic modification* (GM) that yields so-called *genetically modified organisms* (GMO; Uzogara 2000).

One of the most astonishing but equally controversial GM techniques is *transgenesis*, which moves genes around beyond nature's boundaries. For example, genes from an animal can be inserted in a plant, or genes can be transferred between unrelated plant spe-

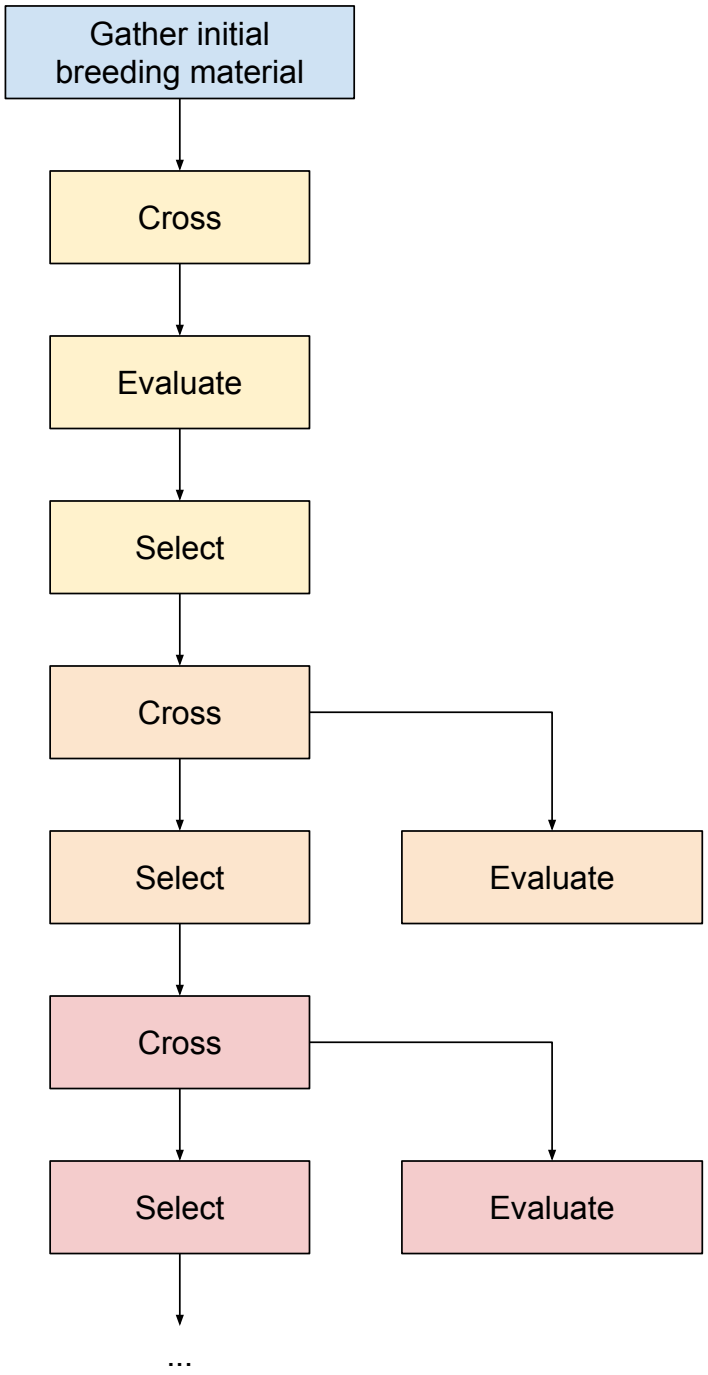


Figure 1.5: Schematic overview of a simple marker-assisted recurrent selection plant breeding scheme using genomic selection. As in a traditional scheme, the first generation consists of crossing, evaluation and selection, in this order. A genomic prediction model is built based on the genotypes and phenotypes of the selection candidates in this first generation, and used for selection. In subsequent generations, selection is performed immediately after crossing using the prediction model that was previously constructed. In parallel, the selection candidates are phenotyped so that the model can be updated to provide more accurate predictions, as the number of individuals with both known genotypes and phenotypes increases.

cies that cannot be crossed. A particular example of a GMO that caused a lot of controversy is *Golden Rice™*. This rice variety was genetically modified by inserting several genes, including one naturally found in daffodils, to produce beta-carotene—as contained in other vegetables like carrots, hence the golden colour—in an attempt to address vitamin A deficiency in developing countries. It is estimated that this nutritional deficiency causes the yearly death of hundreds of thousands of children under the age of five, who are most susceptible. Proponents say that Golden Rice is a quick-fix solution that can save many lives, and should therefore be massively adopted, while opponents claim that there are more sustainable alternatives, that do not compromise food security, avoid a financial dependency on the biotech industry, and address the causes of malnutrition—such as poverty and lack of education—rather than addressing its symptoms only. Fear of unanticipated side effects caused by consuming unnatural GMO's containing foreign genes, and of irreversible contamination of non-GMO varieties through undesired cross-pollination or accidental mixing of seeds, plays an important role in the debate on the acceptance or rejection of genetic modification and transgenesis in particular.

Other GM methods include *cisgenesis*, *mutagenesis* and *gene editing*. In case of cisgenesis, specific genes are artificially transferred between species that could also be crossed—meaning that it is in theory also possible, although not necessarily practical, to obtain the same transfer through conventional breeding. Mutagenesis may be used to trigger mutations, for example through radiation with X-rays or ultraviolet light. Although mutations also occur in nature and those that by chance happen to be beneficial are the driving force of evolution, these are very rare, due to which natural evolution is slow and conventional breeding mostly depends on hybridization (crossing) as its source of variation. Artificially inducing additional mutations provides an alternative source of variability that can be exploited for selection and allows to study the effect of new mutations on various phenotypic traits. Gene editing techniques even go a step further than mutagenesis as they allow to cut, paste and correct specific pieces of DNA in living organisms, instead of just triggering mostly random mutations.

The latest addition to the gene editing toolbox called CRISPR-Cas9 (Doudna and Charpentier, 2014; Hsu et al., 2014) is very promising due to its simplicity and allows to edit genomes with extremely high precision up to changing a single nucleotide. This CRISPR-Cas9 method can be used to alter specific genes, without performing any crosses nor transferring genes from other organisms, offering an efficient alternative to both conventional breeding as well as to older GM methods such as transgenesis and cisgenesis. Moreover,

gene editing has the potential of fixing mutations that cause genetic diseases, also in humans. Remember the example of cystic fibrosis, a serious condition caused by a mutation in a single gene—what if we could simply cut out the mutation and replace it with the correct DNA sequence? Many studies have been initiated with the aim to develop new cures for this and various other diseases, including cancer, although genetic engineering of humans does raise questions similar to and beyond those posed in the context of GMO's. Hardly anyone would oppose against the use of gene editing to try curing serious diseases like cystic fibrosis or cancer, but what about editing human embryos to add abilities such as improved night vision that are currently found in other species? And what about giving parents the ability to determine certain characteristics such as the eye colour of their children? For sure, an ethical debate is needed and scientists agree that the time for this has come as the required technologies have arrived. Gene editing in humans no longer belongs to the realm of science fiction, and many breakthroughs are expected to follow within the next decade.

Going back to plant breeding, it must be noted that due to the controversy GMO's are not yet a global industry, with six countries growing about 95% of all commercial GM crops (USA, Brazil, Argentina, India, Canada and China). More than half of the production takes place in the USA. Most plant breeding companies currently develop their main products without the use of GM, which may then be further augmented for markets where GMO's are allowed.

1.3 OPTIMIZING PLANT BREEDING

Plant breeding is a complex process of which many aspects can still be improved, especially when using the now largely available molecular marker data to make better, more informed decisions. In this thesis we address several optimization problems related to marker-assisted plant breeding. The techniques used to solve these problems are introduced in chapter 2 and some specific methods were implemented generically—i. e. independently from any application—in a Java framework called JAMES, which is presented in chapter 3.

Chapter 4 describes an approach to sample diverse, representative subsets, known as *core subsets*, from large plant collections. Core subset selection was initially introduced in the context of genetic resource conservation, but has many applications in plant breeding as well. For example, breeders may want to compose genetically diverse material to initiate a breeding program, or to phenotype a

representative sample to be used as the training population for a genomic prediction model from which the genetic value of the remaining plants can be estimated. Our software, called Core Hunter, samples cores with complementary allele frequencies or phenotypic trait values to maximize diversity within the core and representativeness of the individual plants from the entire collection.

Next, chapter 5 explores long-term genomic selection strategies that balance gain and diversity. These are needed because previous research showed that genomic selection accelerates the loss of diversity in the breeding population and, as mentioned before, no improvement can be made in the absence of variability. We compare several existing and new diversity management strategies by simulating the genomic selection scheme shown in figure 1.5.

Finally, chapter 6 introduces Gene Stacker: a transparent and flexible crossing scheme generator used to stack several genes, encoding for simple traits, from multiple parents into a single new plant. Gene Stacker is an example of a tool that requires *phase-known* genotype data, to compute the distribution of the offspring of each particular crossing. Therefore, Gene Stacker carefully monitors the linkage phase of all genotypes through the scheme, and avoids as well as reports any ambiguities that may result in an undesired linkage phase. To handle the massive amount of possible crossing schemes we included several heuristics in Gene Stacker that can be enabled to reduce the number of explored schemes, providing a quality-runtime tradeoff that allows to obtain good schemes for complex stacking problems involving up to ten or more genes.

2

INTRODUCTION TO OPTIMIZATION

Optimization problems of all kinds frequently arise in various fields. Just think about daily life: whether you are trying to minimize the time needed to get home from work while picking up your children at school and stopping by the supermarket, figuring out how to put a maximum number of dirty plates in the dishwasher, or organizing your clothes to minimize the time needed to find a matching outfit—these are all optimization problems, for which a number of solutions exist, some of which are preferred over others. Coming up with the best possible solution can be very tough, even more so when trying to find it by hand. Carefully modelling the problem in a mathematical way allows us to use calculus and optimization algorithms to help identify the optimal solution.

An algorithm is a finite sequence of simple instructions that transform some specific input into a desired output. For example, a recipe is an algorithm that describes how to combine several ingredients to prepare a tasty dish. In computer science, algorithms define the sequence of steps taken by a computer to perform a certain task or to solve a certain problem.

2.1 OPTIMIZATION PROBLEMS

A general optimization problem is mathematically formulated as

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && x \in \Omega \end{aligned}$$

where x is a solution, $f(x)$ is a scalar objective function (i. e. assigns a certain value to any given x), and Ω is the so-called feasible solution space. Alternatively, the objective function may need to be maximized, but often only one of both formulations is considered in optimization theory because maximizing $f(x)$ can easily be achieved by minimizing $-f(x)$ and vice versa. In general, a minimum or maximum of $f(x)$ is called an extremum or optimum.

As a simple example, suppose that we want to build a fence attached to one side of a building, enclosing the largest possible rectangular area when using $20\,m$ of fencing material (Dawkins, 2016). The chosen side of the building is more than $20\,m$ long, meaning that

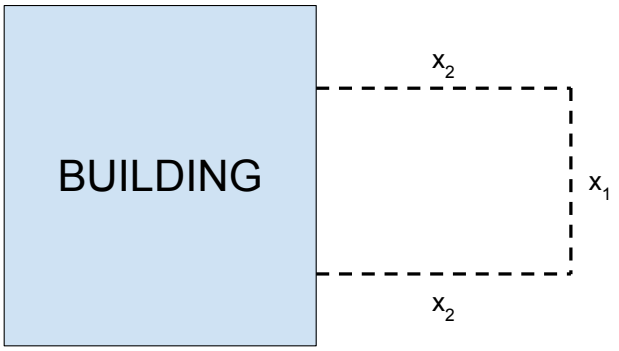


Figure 2.1: Example of maximizing the rectangular area enclosed by a fence attached to a building, using a certain amount of fencing material.

no fencing is needed there in any configuration (figure 2.1). This description leads to the following mathematical formulation:

$$\begin{aligned} \text{maximize} \quad & f(x_1, x_2) = x_1 x_2 \\ \text{subject to} \quad & x_1 \in \mathbb{R}, x_2 \in \mathbb{R} \\ & x_1 \geqslant 0, x_2 \geqslant 0 \\ & x_1 + 2x_2 = 20. \end{aligned}$$

Here, a solution x is defined by two positive real valued variables x_1 and x_2, and the objective function to be maximized corresponds to the area enclosed by the fence. In addition, the total amount of fencing material should equal 20 m, which constrains the feasible values of both variables. We can now find the optimal configuration using basic calculus. First, we combine the objective function and constraint to obtain a function in one variable, either x_1 or x_2—the choice is arbitrary. From

$$x_1 + 2x_2 = 20$$

follows

$$x_1 = 20 - 2x_2$$

and

$$f(x_1, x_2) = x_1 x_2 = (20 - 2x_2)x_2 = 20x_2 - 2x_2^2 = f(x_2).$$

From figure 2.2 we can already see that this function reaches a maximum halfway the feasible range $[0, 10]$, i. e. for $x_2 = 5$. To obtain a formal confirmation of our visual result, we set the derivative of $f(x_2)$ to zero to find its candidate local extrema:

$$f'(x_2) = 20 - 4x_2 = 0 \Leftrightarrow x_2 = 5.$$

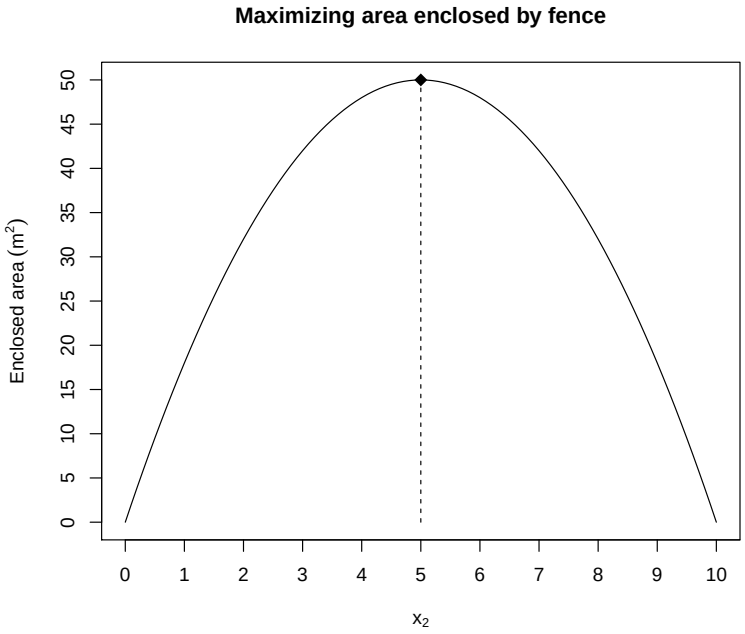


Figure 2.2: Objective function of maximizing the area enclosed by the fence, for $x_2 \in [0, 10]$, which is the feasible range as inferred from the constraints $x_2 \geqslant 0$, $x_1 \geqslant 0$ and $x_1 = 20 - 2x_2$.

Looking at the second derivative $f''(x_2) = -4 < 0$ confirms that this single candidate is an extremum, and more precisely a maximum—not a minimum. Since $f(x_2)$ has only one local extremum, we know that it is also a global extremum, in this case the global maximum, meaning that there does not exist any other solution with a higher value. As the corresponding configuration, with $x_2 = 5$ and $x_1 = 20 - 2x_2 = 10$, falls in the feasible range of both variables, we have found the best solution, enclosing the largest area (50 m^2) using the available fencing material.

The example solved above is a simple *continuous* optimization problem, where all variables take values from an uncountably infinite set such as the real numbers, which allows the use of calculus techniques. Here, we had only two variables, and using the imposed constraint we were able to further reduce the formulation to a basic univariate problem. Also, we were lucky that there was only a single maximum, and that it fell inside the range of feasible values of both variables. Things can get much more complicated as for problems with multiple variables we need to look at partial derivatives and their interactions. Also, in general, there may be several local extrema which often severely complicates the task of finding the global optimum, and we need to make sure that we do not miss the best solution if it happens to fall at the border of the

feasible solution space. Moreover, much more complex constraints may apply than the simple linear constraint in our example.

A huge variety of methods have been developed for specific types of continuous optimization problems, such as for problems without any constraints, those with only equality constraints, and problems with both equality and inequality constraints. Some methods only use function values, while others rely on the availability of first and possibly also second-order derivatives. We do not go further into the topic of continuous optimization here, because the plant breeding problems dealt with in this thesis are *discrete* optimization problems, which require a different approach. The interested reader is referred to Patriksson et al. (2017). Yet, we briefly touch the subject of continuous optimization again in chapter 5.

2.2 DISCRETE OPTIMIZATION

In a discrete optimization problem some or all variables are restricted to discrete values, such as integers or natural numbers, which may further complicate the search for an optimal solution. In general a discrete problem can still have an infinite number of solutions—even when all variables are discrete—but often the values are constrained to a finite set, such as binary values, which when combined in all possible ways yield a finite number of feasible solutions. Such problems are called *combinatorial* optimization problems and can in theory always be solved exactly by simply evaluating all possible solutions, and selecting the best one. Yet, such exhaustive search usually takes way too much time, and therefore more sophisticated techniques are required to solve most combinatorial optimization problems. Still, some algorithms are based on the idea of generating all solutions while taking shortcuts where possible. Therefore, we first describe how to explore the entire solution space through an exhaustive search.

2.2.1 *Exhaustive search*

A common way to generate all solutions of a combinatorial optimization problem is by representing the search space as a *tree* that reflects all combinations of possible variable values. For example, suppose that we want to generate all subsets of a collection of n items. We can formalize this combinatorial problem using n binary variables x_1 to x_n, where $x_i = 1$ means that the i-th item is selected, and $x_i = 0$ means it is not. Figure 2.3 shows how the search space can be represented as a tree, for a collection with $n = 3$ items, in

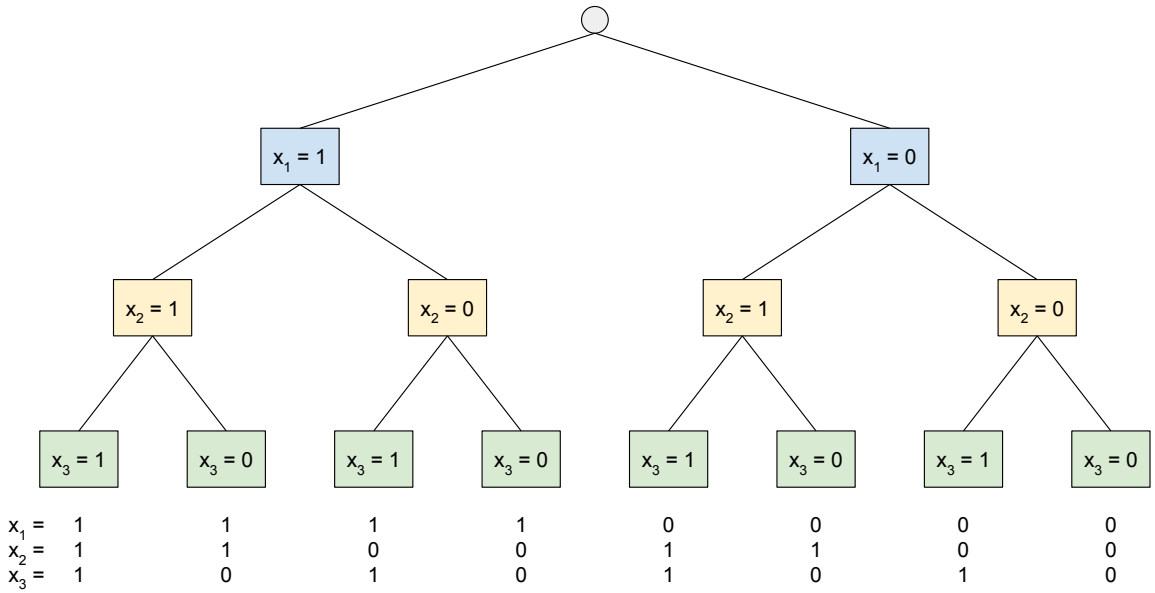


Figure 2.3: Tree representation of the search space when generating all eight subsets of a collection with three items. Each branch of the tree assigns a possible value to a variable, and each leaf (green) corresponds to one of the eight solutions.

which case there are eight possible subsets. Starting at the root of the tree (the small circle) we have not yet assigned any value to any variable x_i. The first thing to decide is whether or not to include the first item in the selection, and both options need to be considered in order to generate all subsets. As such, the tree splits into two *branches* at its root—one with $x_1 = 1$ and the other with $x_1 = 0$. From each of the two obtained blue *nodes* we repeat the same process for the second variable x_2, creating four new branches leading to four new yellow nodes. From left to right, the first yellow node corresponds to the partial solution in which both the first and second item are selected, followed by the two options that only include the first or second item, respectively, and finally a currently empty selection. The remaining decision is whether or not to include the third item, which introduces a total of eight new branches—two from each yellow node—leading to the so-called *leaves* of the tree (green). Each leaf corresponds to one of the eight possible subsets.

To construct the search tree we can either follow a *depth-first* or *breadth-first* approach (figure 2.4). In a depth-first search (top)—a method also known as *backtracking*—immediately after assigning the first possible value to the first variable, $x_1 = 1$, we go further down the tree by considering the first option for the second followed by the third variable as well, i. e. $x_2 = 1$ and subsequently $x_3 = 1$. As such, the third visited node is already a leaf, corresponding to the solution in which all three items are selected. Since there

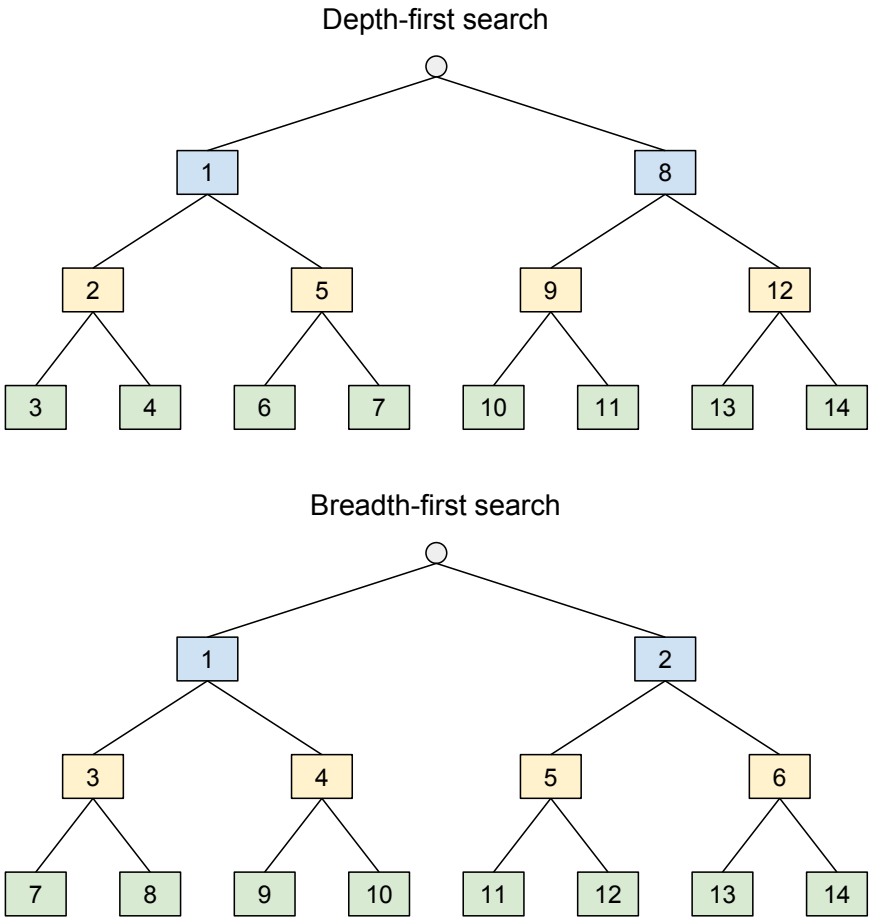


Figure 2.4: Order in which the nodes of the search tree are visited when following a depth-first (top) or breadth-first (bottom) approach.

are no further branches to explore, we go back up one level in the tree, to node two. Here, there is a second, unexplored branch that sets $x_3 = 0$, which again leads to a solution, this time having selected only the first and second item. Going back up, we see that all branches from node two have now been explored and therefore go back up one more level to node one, from which we now follow the second option with $x_2 = 0$, which eventually leads to two new solutions. This process continues until we arrive back at the root and find that there are no more branches to explore, which means that the entire tree has been traversed and that all solutions have been generated.

A breadth-first search (figure 2.4; bottom) simply visits the nodes of the tree level by level. Although this order may seem more intuitive, a depth-first search is often easier to program and—when done carefully—also more efficient. For example, a breadth-first search requires to make copies of partial solutions at each node, because all possible extensions are simultaneously considered. In

practice, this is implemented with a queue that contains all generated partial solutions that still need to be further extended. At the root node, an initial partial solution in which no variable values have yet been assigned is added to the queue. Then, as long as there are elements in the queue, the first one is dequeued and all possible extensions of the respective partial solution are added at the end of the queue. Repeating this process until the queue is empty generates all possible solutions in a breadth-first order. In contrast, a depth-first search can work with a single current solution that is repeatedly modified while traversing the tree. When following a branch downwards, the current solution is extended by assigning the chosen value to the respective variable. When going back up—the backtracking part—the previously assigned value is erased and ready to be replaced with the next possible value, if any. Because of this advantage, a depth-first approach usually requires much less memory than a breadth-first search. Moreover, the leaves of the tree are visited earlier by a depth-first search, meaning that this approach more quickly generates complete solutions. Pruning criteria used to reduce the number of explored solutions, as explained below, often use information about the best solution that was generated so far. As such, a depth-first search often allows earlier pruning leading to shorter execution times. Yet, for some problems a breadth-first search may be preferred due to specific requirements, or a depth-first search may simply not be possible.

In general, a collection of n items has 2^n possible subsets. Due to this exponential growth, which is typical for a combinatorial optimization problem, it quickly becomes infeasible to generate all solutions. Suppose that we have a reasonably fast computer that can evaluate about one million subsets per second. Table 2.1 indicates how much time would approximately be needed to evaluate all subsets of a collection of varying size. In case of $n = 20$ items, there are about one million subsets, so these can all be evaluated in as little as one second. Doubling the size to $n = 40$ already severely increases the computation time to about 13 days, and evaluating all subsets of a collection with $n = 80$ items would require a truly astonishing amount of time—over 38 billion years, which is almost three times the age of the universe. Even if we would use a high performance computing cluster with hundreds of cooperating machines, it would take millions of years to inspect all possible subsets of a quite small dataset with only 80 items. Therefore, techniques are required to reduce the number of evaluated solutions, for example by pruning branches of the search tree of which we can predict that they cannot lead to an optimal solution.

Items (n)	Subsets (2^n)	Approximate time
20	$\approx$ 1 million	1 second
40	$\approx 10^{12}$	13 days
60	$\approx 10^{18}$	$>$ 36 thousand years
80	$\approx 10^{24}$	$>$ 38 billion years

Table 2.1: Approximate computation time needed to evaluate all subsets of a collection of varying size. It is assumed that about one million solutions are evaluated per second.

2.2.2 *Pruning criteria*

When solving a combinatorial optimization problem we actually only want to find the best solution—not all solutions—and the fewer solutions that need to be explored, the better. One approach to reduce the number of solutions constructed in an exhaustive search, and as such its execution time, is by incorporating pruning criteria that skip certain parts of the search tree. Of course, to still guarantee optimality, we can only skip those branches of which we know that they cannot lead to an optimal solution.

For example, consider a subset selection problem known as the *knapsack* problem. Given n items that each have a certain profit and weight, we want to select a set of items with the highest possible total profit, taking into account that their total weight cannot exceed a certain capacity. To solve this problem we could generate all subsets, discard those that violate the capacity constraint, and from the remaining solutions take the one with the highest profit. However, we can easily prune the search space, for example by taking into account that adding more items to the selection can only increase the total weight. Therefore, if a partial selection already exceeds the capacity, there is no need to consider any further branches from the corresponding node in the search tree. Furthermore, we can try to predict the maximum profit that can be achieved by extending a given partial solution. If this upper bound is lower than the profit of the best solution found so far, we need not further extend the respective partial solution, since it can never lead to a solution that outperforms the currently known best solution.

A trivial bound for the knapsack problem would be to take the profit of the current selection increased by the total profit of all items for which no decision has yet been made, as obviously we can impossibly obtain a higher total profit than in case we select all these remaining items. Although this bound may severely overes-

timate the maximum obtainable profit—and much better alternatives have been developed that also take into account the capacity of the knapsack—many branches would already be skipped with this simple criterion. A pruned generation algorithm that applies upper bounds, or lower bounds in case of a minimization problem, is called a *branch-and-bound* algorithm. To prune as many branches as possible, we need tight bounds that fully exploit the characteristics of the specific problem at hand.

2.2.3 *Problem complexity*

Even when pruning the search tree where possible, an exponentially growing number of solutions typically needs to be explored to solve a combinatorial optimization problem with an approach based on exhaustive enumeration. Effective pruning criteria and bounds may significantly reduce the execution time, and as such allow to solve larger problems within reasonable time, but sooner or later we often still hit a computational limit. Luckily, many combinatorial optimization problems can be solved without the need to build a search tree that generates an exponentially increasing number of solutions. For example, consider a special case of the knapsack problem where all items have equal weight $w_i = 1$. Maximizing the total profit with a capacity of k is then easily accomplished by selecting the k items with the highest individual profit.

Efficient algorithms have been developed for many specific optimization problems—yet, others showed to be exceptionally difficult. For example, nobody has been able to find an algorithm that can solve the general knapsack problem described above, without in the worst case evaluating an exponential number of solutions. It is believed that the knapsack problem cannot be solved "efficiently", which leads us to one of the most important open questions in computer science: the P versus NP problem.

Every optimization problem has a corresponding *decision* problem that asks the question (formulated here for maximization):

> Is there any solution $x \in \Omega$ that has a value $f(x) > v$?

where v is the input of the decision problem. For example, we can formulate the knapsack problem as a decision problem by asking:

> Is there any selection with total profit at least v that does not exceed the imposed capacity?

An optimization problem is always at least as hard as its corresponding decision problem because if we can efficiently find the optimum, we can also efficiently solve the decision problem by comparing the

given value v with that of the optimal solution. A decision problem that can be solved efficiently, more precisely in polynomial time, is said to belong to a class called P. This means that an algorithm is known that solves the problem with at most cn^k operations, where c and k are arbitrary constants, and n is the size of the input. Obviously, an exhaustive search or branch-and-bound algorithm does not run in polynomial time as it may in the worst case evaluate an exponential number of c^n solutions. More generally, the class NP contains those decision problems for which an answer can be verified—but not necessarily obtained—in polynomial time. For example, in case of the knapsack problem, verifying whether a given selection stays within the capacity, and whether its value is indeed larger than a given value v, is as easy as summing up the weights and profits of the selected items and comparing these to a certain value. Thus, the knapsack problem is contained in NP. It is clear that $P \subseteq NP$, i. e. every problem in P is also in NP, since finding a solution is always at least as hard as verifying one.

A million dollar prize is rewarded by the Clay Institute to the one who can prove whether or not $P \neq NP$. Up to the challenge? Take a look at https://goo.gl/IUGWVz. But be warned, many talented minds have tried before you, without success.

The question remains however whether $P = NP$, i. e. whether all problems whose solutions are easily verified can also be solved efficiently. Although it is generally believed that $P \neq NP$, no one has been able to prove it (Fortnow, 2009). Still, there are so many problems in NP, such as the knapsack problem, for which nobody has yet found a polynomial algorithm, that it would be truly remarkable if it turns out that $P = NP$. Such a result would have serious implications in many fields, including cryptography. The strength of certain encryption algorithms depends on the generally accepted assumption that $P \neq NP$ as, for example, it should be easy to quickly encrypt and decrypt a confidential file or message, but extremely difficult for a potential intruder to reveal its contents without having access to the applied encryption key (such as a password).

Certain problems in NP are at least as difficult as all others—these are referred to as NP-complete problems. If a polynomial algorithm would be discovered for any NP-complete problem, it could be used to efficiently solve all other problems in NP as well. Therefore it is generally assumed that NP-complete problems cannot be solved efficiently, since they are the problems in NP that are most likely not in P. The knapsack problem, for example, is NP-complete.

It must be noted that the term "efficient" is used here from a theoretical viewpoint, always meaning "in polynomial time". Even a problem in P can be hard to solve in practice within reasonable time, for example if the best known algorithm takes n^{10} steps and could thus hardly be called efficient for practical purposes, being useless for large inputs. On the other hand, for many NP-complete problems, including the knapsack problem, state-of-the-art algorithms

Algorithm 2.1 Greedy heuristic for the knapsack problem.

Input: n items with profits p_i and weights w_i; capacity W
Output: profitable selection of items with total weight at most W

```
1: sort the items on decreasing ratio of profit per weight (p_i/w_i)
2: start with an empty selection
3: w ← 0
4: for i from 1 to n do
5:     if w + w_i ⩽ W then
6:         add i-th item to selection
7:         w ← w + w_i
8:     else
9:         skip i-th item
10:    end if
11: end for
12: return selection
```

can solve certain practical cases efficiently—yet, in general, with an exponential worst case execution time.

2.2.4 *Heuristics and metaheuristics*

In practice, approximation algorithms are often used to obtain good solutions within reasonable time for tough optimization problems, such as those whose corresponding decision problem is NP-complete. Such inexact algorithms are called *heuristics* and do not guarantee that the result is truly optimal—trading solution quality for execution time. Still, intelligent heuristics can often quickly find near-optimal results, that may be sufficient to deal with the problem at hand in a practical setting.

For example, algorithm 2.1 provides a simple heuristic for the general knapsack problem. We describe this optimization algorithm using *pseudocode*, which defines its outline in a way that is clear and structured but yet independent of any programming language. Pseudocode should be specific enough to allow a programmer to implement the algorithm in his or her favourite language (such as C or Java). In our pseudocode we use the arrow symbol ($\leftarrow$) as a notation for the assignment of a certain value, specified at the right side of the arrow, to the variable stated at the left side of the arrow.

The presented so-called *greedy* heuristic for the knapsack problem is very fast as it just repeatedly selects the remaining item with the highest profit per weight, unless the capacity would be exceeded, but does not guarantee that the best solution will be found.

Some heuristics—known as *metaheuristics*—provide general purpose approximation strategies to explore the solution space in an intelligent way, independently from a specific problem description. A major advantage of using such high-level heuristics is the inherent flexibility that for example allows to solve multiple related problems without changing the optimization engine.

Local search

In particular, *local searches* are metaheuristics that repeatedly modify a given initial solution in an attempt to improve it, until a certain condition is satisfied. Often, the initial solution is randomly generated, or the result of another approximation algorithm. Local modifications are usually formalized through the concept of a *neighbourhood* function that, given a solution, yields a set of similar solutions.

RANDOM DESCENT One of the most basic local search strategies—referred to as *stochastic hill-climbing* or *random descent*—is described in algorithm 2.2 (formulated for a maximization problem). This metaheuristic takes an initial, e. g. randomly generated solution and then iteratively evaluates a randomly chosen neighbour to see if it improves over the current solution. If so, this neighbour replaces the current solution. This process is repeated until a certain stop condition is satisfied, such as a maximum number of iterations, maximum runtime, or maximum time or number of steps without finding any further improvement over the current solution.

Algorithm 2.2 Random descent.

Input:
- objective function $f(x)$
- initial solution $x \in \Omega$
- neighbourhood function $N(x) : \Omega \to \mathcal{P}(\Omega)$
- stop condition

Output: best found solution $x^* \in \Omega$

1: **repeat**
2: pick random neighbour $x' \in N(x)$ of current solution x
3: **if** $f(x') > f(x)$ **then**
4: $x \leftarrow x'$ (accept x' as new current solution)
5: **else**
6: retain x as current solution
7: **end if**
8: **until** stop condition satisfied
9: **return** x

For a solution $x \in \Omega$, the neighbourhood function $N(x) : \Omega \to \mathcal{P}(\Omega)$ yields a set of similar solutions (neighbours). Here, $\mathcal{P}(\Omega)$ refers to the *power set* of Ω, i. e. the set of all subsets of Ω. The neighbourhood function, and the way in which an initial solution is obtained, are problem specific. For example, for the knapsack problem, a possible neighbourhood function could be one that randomly adds or removes an item to/from the selection, while ensuring that the capacity is not exceeded:

$$N(x) = \{x' \mid w(x') \leqslant W \wedge \exists i : x'_i = 1 - x_i \wedge \forall j \neq i : x'_j = x_j\}.$$

According to this definition, $N(x)$ contains all solutions x' with weight $w(x')$ below the capacity W that differ from the current solution x in a single variable, whose value is changed from 0 to 1 (addition) or from 1 to 0 (removal). An initial valid solution can easily be obtained, for example by starting with an empty selection or by taking a random selection and then, if necessary, removing some items to bring the total weight below the allowed capacity.

Given that a sufficient number of iterations are performed, the random descent heuristic will converge towards a local optimum, i. e. a solution x for which there exists no neighbour $x' \in N(x)$ with $f(x') > f(x)$. Yet, this does not necessarily mean that a global optimum has been obtained. For example, imagine you would like to find the highest point on earth, starting from some location, by walking in a random direction that leads upwards. If you walk long enough you will eventually arrive at the top of some hill or mountain, but chances are low that you will find yourself at the summit of Mount Everest. For the same reason, the random descent algorithm will likely not yield the best possible solution. Sometimes we may want to walk down a bit to look for an even higher mountain than the one we just climbed, but the random descent algorithm does not allow such inferior moves. In particular, when using the example neighbourhood described above for the knapsack problem, no removals will ever be accepted as these always decrease the total profit. However, it may be beneficial to remove an item so that it can subsequently be replaced by other, potentially more valuable items. Of course we can not expect that a heuristic always finds the optimal solution but we do want to approximate this optimum as closely as possible. Therefore, many more intelligent and more powerful local searches have been developed.

The simulated annealing algorithm is based on the annealing process in metallurgy. To be able to shape metal it is heated to a very high temperature that provides the atoms with sufficient energy to move around freely. The material is then slowly cooled to obtain a strong product.

SIMULATED ANNEALING One popular extension of the basic random descent algorithm is known as *simulated annealing* (Kirkpatrick et al., 1983). This method may accept worse neighbours as the new current solution to escape from local optima. The probability of accepting such inferior solutions decreases over time, so that initially

Algorithm 2.3 Simulated annealing.

Input:
- objective function $f(x)$
- initial solution $x \in \Omega$
- neighbourhood function $N(x) : \Omega \to \mathcal{P}(\Omega)$
- temperature function $t(i)$
- acceptance function $p(\Delta, t)$
- stop condition

Output: best found solution $x^* \in \Omega$

1: $i \leftarrow 0$
2: **repeat**
3: pick random neighbour $x' \in N(x)$
4: compute $\Delta \leftarrow f(x') - f(x)$
5: with probability $p(\Delta, t(i))$: set $x \leftarrow x'$
6: $i \leftarrow i + 1$
7: **until** stop condition satisfied
8: **return** x

there is a lot of freedom but eventually the search will converge to an optimum—hopefully a global optimum or a close approximation. This gradual reduction of freedom is modelled through a temperature, that determines the probability to accept a worse solution and decreases over time. The simulated annealing method is described in algorithm 2.3 (again for a maximization problem).

The temperature function $t(i)$ provides the (positive) temperature of the optimization engine in the i-th iteration and is usually chosen to always decrease and approach zero after a large number of iterations. One example is the function

$$t(i) = T_0 \lambda^{\lfloor i/n \rfloor}$$

where T_0 is the initial temperature, n is an integer that controls the number of subsequent steps with the same temperature, and $0 < \lambda < 1$ a parameter that determines the cooling rate. The acceptance function $p(\Delta, t)$ defines the probability to accept a difference of $\Delta = f(x') - f(x)$ for the objective function value when moving from solution x to x'. It is usually defined as

$$p(\Delta, t) = \begin{cases} 1 & \text{if } \Delta > 0 \\ e^{\Delta/t} & \text{else} \end{cases} \tag{2.1}$$

which means that better solutions, i. e. with $f(x') > f(x)$ and thus $\Delta > 0$, are always accepted, and that the probability to accept an inferior neighbour, with $\Delta \leqslant 0$, exponentially decreases for a larger

difference in objective function value. Moreover, a worse solution will more likely be accepted if the current temperature is high.

The temperature and acceptance function, together with the applied neighbourhood, largely define the behaviour of the simulated annealing algorithm, and should be fine-tuned for each specific application, for example by trying different functions and parameter values and comparing the results.

PARALLEL TEMPERING *Replica exchange Monte Carlo search* or *parallel tempering* (Earl and Deem, 2005; Thachuk et al., 2009) is another advanced local search metaheuristic based on the same principles as simulated annealing. Here, instead of decreasing the temperature over time, multiple *replicas* with a different fixed temperature are executed in parallel. From time to time these cooperating subsearches may exchange their current solution to push the best solutions to the coolest replicas for convergence, and the worst solutions towards the hottest replicas to be able to escape from local optima. The parallel tempering method is described in algorithm 2.4 (as before, for a maximization problem).

The k replicas each have their own initial solution, and are assigned unique, equally-spaced, and increasing temperatures in the range $[t_{min}, t_{max}]$. At first, the highest-quality initial solution specified for any of the replicas is taken as the global best solution. Next, to improve this global approximation of the optimum, the parallel tempering algorithm repeatedly executes its search loop that consists of two main phases.

First, n iterations are performed for each replica, following the same procedure as simulated annealing but with its fixed temperature, and usually with the default acceptance function $p(\Delta, t)$ as defined in equation (2.1). In addition, the best solution found so far across all replicas is tracked, which is eventually returned as the final solution of the main search. Because the steps taken by each individual replica are independent, it is possible to execute these in parallel on modern computers with multi-core architectures—a major benefit of the parallel tempering algorithm as this allows to obtain better solutions with a limited increase in execution time.

Secondly, the current solutions of adjacent replicas r and $r+1$ are considered to be swapped, based on a swap function $q(\Delta_r, t_r, t_{r+1})$ that is usually defined as

$$q(\Delta_r, t_r, t_{r+1}) = \begin{cases} 1 & \text{if } \Delta_r > 0 \\ e^{(\frac{1}{t_r} - \frac{1}{t_{r+1}})\Delta_r} & \text{else} \end{cases}$$

Algorithm 2.4 Parallel tempering.

Input:
- objective function $f(x)$
- desired number of search replicas (k)
- series of k initial solutions $x_i \in \Omega$
- neighbourhood function $N(x) : \Omega \to \mathcal{P}(\Omega)$
- temperature range $[t_{min}, t_{max}]$
- acceptance function $p(\Delta, t)$
- swap function $q(\Delta, t_1, t_2)$
- number of replica steps per iteration (n)
- stop condition

Output: best found solution $x^* \in \Omega$

for i from 1 to k **do**
  $t_i \leftarrow t_{min} + \frac{i-1}{k-1}(t_{max} - t_{min})$
end for
$x_{best} \leftarrow \mathrm{argmax}_{1 \leqslant i \leqslant k} f(x_i)$
$s \leftarrow 0$
repeat
  for i from 1 to k **do**
    repeat n times
      pick random neighbour $x_i' \in N(x_i)$
      compute $\Delta_i \leftarrow f(x_i') - f(x_i)$
      with probability $p(\Delta_i, t_i)$: set $x_i \leftarrow x_i'$
      if $f(x_i) > f(x_{best})$ **then**
        $x_{best} \leftarrow x_i$
      end if
    end repeat
  end for
  $r \leftarrow s + 1$
  while $r < k$ **do**
    compute $\Delta_r \leftarrow f(x_{r+1}) - f(x_r)$
    with probability $q(\Delta_r, t_r, t_{r+1})$: swap x_r and x_{r+1}
    $r \leftarrow r + 2$
  end while
  $s \leftarrow 1 - s$
until stop condition satisfied
return x_{best}

with $\Delta_r = f(x_{r+1}) - f(x_r)$. This means that, if the current solution of replica $r+1$ has a better objective function value than that of the r-th replica, these solutions are always swapped to push the best solutions to the coolest replicas and vice versa, for reasons explained above. In addition, similar to the probabilistic acceptance of inferior neighbours, swaps that push solutions in the opposite direction may also be performed—yet with a probability that decreases for a larger difference in objective function value or a larger difference in replica temperature.

An auxiliary variable s, whose value alternates between 0 and 1, is used to ensure that, in a single step, solutions can only migrate to an adjacent replica. More precisely, in odd iterations of the main algorithm, swaps are only considered between replicas 1 and 2, 3 and 4, 5 and 6, and so on. On the other hand, even iterations only allow swaps between replicas 2 and 3, 4 and 5, etc. This approach ensures a gradual temperature change for each current solution x_i as in the simulated annealing algorithm.

It is important to note that the parallel tempering algorithm—unlike simulated annealing—incorporates a continuous source of new variation provided by the hot replicas, which avoids potential issues such as premature convergence when for example using a temperature function that cools too rapidly. Therefore, it may be easier to fine-tune the parameters of parallel tempering as compared to simulated annealing. Furthermore, as each replica starts with its own initial solution, the parallel tempering algorithm has a built-in multi-start feature that may on its own already significantly improve the value of the obtained global solution.

Population-based metaheuristics

Another major class of metaheuristics besides local searches are population-based approaches such as *evolutionary algorithms*—the most popular ones being *genetic algorithms* (GA; Holland 1975). The idea behind GA is to mimic natural or artificial selection in order to improve a population of initial (for example randomly generated) solutions towards a global optimum—much like how breeders improve plant populations for a trait of interest through repeated crossing and selection. Usually, solutions are represented as character strings to which a certain fitness is assigned based on the corresponding objective function value, similar to a genotype encoding for observable phenotypes. In every iteration several solutions are selected as parents to be recombined into new solutions through crossover of their corresponding string representation, some modifications (called mutations, following biological terminology) are

possibly applied to these new solutions, and certain solutions are discarded. The highest-quality solutions have the highest probability to be parents and to survive to the next generation, mimicking Darwin's survival of the fittest. After a large number of iterations, the best solution of the population is returned as the final approximation of the optimum.

Other population-based methods, such as *particle swarm optimization* (Kennedy, 2011) and *differential evolution* (Storn and Price, 1997), are particularly suited for continuous optimization as they naturally represent solutions as vectors of real numbers. Again, a population of solutions is maintained that together move towards a global optimum by iteratively combining characteristics of already obtained high-quality solutions in a clever way.

In general, population-based algorithms have the advantage of providing a more global exploration of the solution space as compared to local searches. On the downside they are also more computationally demanding which makes them particularly useful to deal with problems where for example many local optima of highly varying quality are expected, in which case simpler and faster techniques may not yield sufficiently good solutions. There is always a tradeoff between execution time and solution quality when using heuristics and the options are endless—several techniques can even be combined into a hybrid heuristic such as a genetic algorithm with a local search as mutation operator. When looking for an appropriate optimization algorithm among the huge amount of possibilities, simple methods should not be forgotten, and the added value of more complex and slower techniques should be carefully evaluated.

For the problems addressed in this thesis there was no need to use population-based algorithms. Therefore, we do not provide details here and refer the interested reader to the references mentioned above and the excellent *Handbook of Metaheuristics* (Gendreau and Potvin, 2010) which also contains descriptions and guidelines for many local searches.

2.3 MULTI-OBJECTIVE OPTIMIZATION

Optimization problems often have multiple competing objectives, in which case it is not immediately clear what it means for a solution to be *optimal*. Suppose for example that we want to construct an efficient crossing scheme to create a new plant variant. Efficiency could e.g. be measured as the number of required selection cycles, reflecting time, or the total number of plants that need to be grown and screened for selection throughout the entire scheme, reflecting

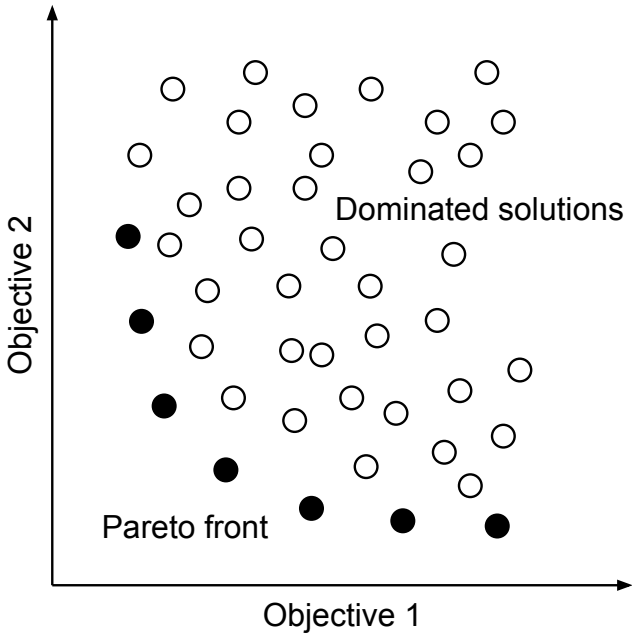


Figure 2.5: Illustration of a Pareto front for two objective functions that are both minimized. Here, the set of feasible solutions Ω is finite and depicted with circles in the objective space. Filled circles correspond to the values of Pareto optimal solutions, while the remaining circles reflect dominated solutions.

cost and required resources. Ideally we may want to minimize both of these measures but likely reducing the available time will lead to higher costs as this requires to make more progress per cycle, i. e. this asks for larger population sizes to be able to select rare offspring of exceptionally high quality. Therefore, we settle to find a good balance between the two objectives. In case a certain solution can not be improved in terms of a single objective without deteriorating at least one other objective it is said to be *Pareto optimal*. The possibly infinite set of Pareto optimal solutions forms a so-called *Pareto front* in the objective space (figure 2.5) and reflects optimal tradeoffs between the different objectives.

A multi-objective optimization problem can be formulated as

$$\begin{aligned} &\text{minimize} && [f_1(x), f_2(x), \ldots, f_n(x)] \\ &\text{subject to} && x \in \Omega \end{aligned}$$

where $f_i(x)$, $i = 1, \ldots, n$ $(\geqslant 2)$, are scalar objective functions and Ω is, as before, the feasible solution space. A solution $x^* \in \Omega$ is called Pareto optimal if there does not exist any other solution $x' \in \Omega$ with (a) $\forall i : f_i(x') \leqslant f_i(x^*)$; and (b) $\exists j : f_j(x') < f_j(x^*)$. If such other solution x' does exist, it is said to dominate x^*, as x' is strictly better than x^* in terms of some objective while still being at least as good for all other objectives. Similarly, a solution

$x^* \in \Omega$ is called *weakly* Pareto optimal if there exists no other solution $x' \in \Omega$ with $\forall i : f_i(x') < f_i(x^*)$, i. e. there does not exist any solution that is better for all objectives. Each solution evaluates to a vector of n objective function values, and the Pareto front is the set of those vectors corresponding to all Pareto optimal solutions, describing the tradeoffs between the individual objectives. Solving a multi-objective optimization problem usually means that we want to determine one or a few Pareto optimal solutions satisfying our preferences.

2.3.1 *Weighted index and normalization*

One way to obtain Pareto optimal solutions of a multi-objective optimization problem is through linear scalarization, also known as the weighted sum method. Here, the problem is converted into a single-objective optimization problem by minimizing a weighted index that combines all objectives:

$$\begin{aligned} &\text{minimize} \quad F(x) = \sum_{i=1}^{n} w_i f_i(x) \\ &\text{subject to} \quad x \in \Omega \end{aligned}$$

with $\forall i : w_i \geqslant 0$ and often $\sum_{i=1}^{n} w_i = 1$. Weights can either be determined a priori to directly reflect preferences, or multiple Pareto optimal solutions can be obtained by applying different weights followed by an a posteriori selection. For each choice of strictly positive weights an optimum of the weighted sum is Pareto optimal for the corresponding multi-objective problem. If one or more weights are zero, a weakly Pareto optimal solution may be produced. Unfortunately, however, Pareto optimal solutions located at non-convex regions of the Pareto front, if any, can not be obtained through a specific choice of weights (Marler and Arora, 2010). Still, the weighted sum method is often used due to its simplicity, although in practice it may not always be easy to determine weights that accurately model preferences. One key insight is that when weights are set to reflect relative importance of the objective functions, the latter should be normalized. Else, some objectives may naturally dominate the sum, which makes the process of setting desirable weights somewhat arbitrary.

A possible normalization strategy is to linearly rescale the i-th objective function $f_i(x)$ from its original range $[l_i, u_i]$ to $[0, 1]$:

$$f_i'(x) = \frac{f_i(x) - l_i}{u_i - l_i}.$$

There are several options to determine upper and lower bounds u_i and l_i, respectively, for each objective function. When $f_i(x)$ is minimized, it is evident to use (an approximation of) the minimum of this individual objective function as its lower bound:

$$l_i = \min_{x \in \Omega} f_i(x).$$

Similarly, we could use its absolute maximum as an upper bound but this does not necessarily reflect the range of values for Pareto optimal solutions, as there may be solutions with a much higher value for $f_i(x)$ than the worst one observed along the Pareto front. Therefore, in a multi-objective optimization setting where the primary goal is to identify good tradeoffs between the optimal solutions in terms of the individual objective functions, it is more appropriate to use the *Pareto maximum* as upper bound (Marler and Arora, 2005):

$$u_i = \max_{1 \leqslant j \leqslant n} f_i(x_j^*) \quad \text{with} \quad x_i^* = \operatorname{argmin}_{x \in \Omega} f_i(x).$$

To obtain the normalization ranges of all objective functions we thus first independently find or approximate the individual minima

$$x_i^* = \operatorname{argmin}_{x \in \Omega} f_i(x)$$

for each function $f_i(x)$. Then the ranges are determined as

$$\begin{aligned} l_i &= f_i(x_i^*) \\ u_i &= \max_{1 \leqslant j \leqslant n} f_i(x_j^*). \end{aligned}$$

In case the i-th objective function is to be maximized, the Pareto minimum and absolute maximum of $f_i(x)$ are used as lower and upper bound, respectively, which is equivalent to applying the above formulation when minimizing $-f_i(x)$.

2.3.2 *Pareto front generation*

The weighted sum method described above yields a single (potentially weakly) Pareto optimal solution for each particular choice of weights. In situations where it is difficult to determine appropriate weights a priori, one may choose to generate multiple solutions by varying the weights, followed by an a posteriori selection among the generated Pareto optimal solutions. Yet, as mentioned before, the weighted sum method may not be able to produce solutions representing the complete Pareto front. This issue can be resolved by using alternative methods that were specifically developed to obtain an even representation of the entire Pareto front.

In particular, population-based metaheuristics can be intuitively adjusted to approximate the Pareto front of a multi-objective optimization problem. Instead of pushing the entire population towards

an optimum of a single objective function, the population is then manipulated so that it converges towards an even representation of the Pareto front. Two state-of-the-art examples of multi-objective metaheuristics are the *strength Pareto evolutionary algorithm* (SPEA2; Zitzler et al. 2001) and the *non-sorting genetic algorithm* (NSGA-II; Deb et al. 2002). For combinatorial multi-objective optimization problems in particular, with a finite number of Pareto optimal solutions, it is possible to construct the full Pareto front—for example through an exhaustive or branch-and-bound search that keeps track of the set of currently obtained non-dominated solutions, which converges towards the Pareto front. Such an approach is of course only feasible if there are relatively few Pareto optimal solutions.

Other available techniques to generate a representative set of Pareto optimal solutions include the *directed search domain* method (Erfani and Utyuzhnikov, 2011), *successive Pareto optimization* (Mueller-Gritschneder et al., 2009), *normal boundary intersection* (Das and Dennis, 1998; Motta et al., 2012), and the *normal constraint* method (Messac and Mattson, 2004).

2.4 APPLICATION TO PLANT BREEDING PROBLEMS

Due to the generality of metaheuristics large parts of their implementation can be reused across different applications. Therefore, we implemented random descent, parallel tempering, and many other local searches in a generic Java framework called JAMES (Java metaheuristics search). We introduce JAMES in chapter 3 and apply the included local search techniques for core subset selection in chapter 4, and to balance gain from genomic selection with maintaining population diversity in chapter 5. Both of these applications use the weighted sum method and the normalization procedure described above to balance multiple objective functions.

In chapter 6 we describe the Gene Stacker algorithm that uses a breadth-first search algorithm to generate crossing schemes to stack several genes from multiple parents into a single new individual, with a minimum number of generations, cost, and linkage phase ambiguity. We incorporated several bounds and other pruning criteria to solve complex stacking problems with up to ten or more genes within reasonable time. Many of these pruning criteria are heuristics, meaning that although Gene Stacker is based on an exhaustive enumeration, it is not an exact algorithm as it may happen that the best solution is missed due to the heuristic pruning.

Gene Stacker approximates the entire Pareto front including all tradeoffs between the three optimization objectives, which in this

case generally consists of one or a few schemes per considered number of generations only. The decision maker can then choose the most desirable scheme from the set of provided alternatives, possibly also taking into account additional criteria that are not modelled by Gene Stacker.

Part II

THE JAMES FRAMEWORK

This part describes JAMES: a Java framework for discrete optimization using local search metaheuristics. The JAMES framework provides many generic optimization algorithms that can be applied to various problems by plugging in only the application specific components, such as the objective function that is to be optimized, an appropriate neighbourhood function, and the required search parameters. We use JAMES in subsequent chapters to solve multiple optimization problems related to marker-assisted plant breeding.

3

JAMES: A JAVA METAHEURISTICS FRAMEWORK

SUMMARY

This chapter describes JAMES (v1.1): an object-oriented Java framework for discrete optimization using local search algorithms, that exploits the generality of such metaheuristics by clearly separating search implementation and application from problem specification. A wide range of generic local searches are provided, including (stochastic) hill-climbing, tabu search, variable neighbourhood search and parallel tempering. These can be applied to any user-defined problem by plugging in a custom neighbourhood for the corresponding solution type. Using an automated analysis workflow, the performance of different search algorithms can be compared in order to select an appropriate optimization strategy. Implementations of specific components are included for subset selection, such as a predefined solution type, generic problem definition and several subset neighbourhoods used to modify the set of selected items. Additional components for other types of problems (e. g. permutation problems) are provided through an extensions module which also includes the analysis workflow. In comparison with existing Java metaheuristics frameworks, that mainly focus on population-based algorithms, JAMES has a much lower memory footprint and promotes efficient application of local searches by taking full advantage of move-based evaluation. Releases of JAMES are deployed to the Maven Central Repository so that the framework can easily be included as a dependency in other Java applications. The project is fully open source and hosted on GitHub. More information can be found at `http://www.jamesframework.org`.

GitHub is a popular online code sharing platform built on top of a version control system called `git`. It makes it easy to share code and collaborate remotely, and tracks the full history of changes. More info on: `https://www.github.com`.

3.1 INTRODUCTION

As discussed in section 2.2.3 many optimization problems are difficult to solve, e. g. due to NP-completeness, in which case exact techniques are often not applicable. A common practical approach to deal with this issue is to use inexact algorithms that find valuable approximations of the best solution within reasonable time. For this purpose, metaheuristics are frequently applied, with the

major advantage that they can be adjusted easily to solve various optimization problems arising from different fields, i. e. with the addition of only the necessary problem specific components such as neighbourhood functions in case of a local search, or crossover, mutation and selection operators in case of a genetic algorithm. In this context, software frameworks are valuable tools to reduce the effort needed to apply well-established metaheuristics to newly defined problems. Such frameworks are also helpful for the implementation of new ideas and comparison with existing algorithms, and to create hybrid combinations of different search techniques.

Parejo et al. (2012) provide an overview of metaheuristics frameworks that have been developed over the last few decades, each targeting a certain class of algorithms and/or specific type of applications, implemented in a variety of object-oriented programming languages such as C++, C# and Java. For example, ParadisEO (Cahon et al., 2004) is an extensive C++ framework that supports both single- and multi-objective optimization using local search and population-based metaheuristics, with extensions for parallel and distributed computation. Other options for C++ users include EasyLocal++ (Di Gaspero and Schaerf, 2003) and MALLBA (Alba et al., 2007).

Most Java frameworks focus on population-based algorithms and especially evolutionary algorithms, including JCLEC (Ventura et al., 2008), ECJ (White, 2012), EvA2 (Kronfeld et al., 2010), Opt4j (Lukasiewycz et al., 2011), OAT (Brownlee, 2007) and jMetal (Durillo and Nebro, 2011). Some frameworks excel for specific types of evolutionary algorithms. For example, ECJ is widely used within the field of genetic programming. Other frameworks, like jMetal, mainly target multi-objective optimization for which specific population-based metaheuristics like NSGA-II (Deb et al., 2002) and SPEA2 (Zitzler et al., 2001) have been developed. These are all computationally demanding techniques that might not be needed when dealing with single-objective problems of moderate complexity, for which simpler local search based methods may perform well enough. To our knowledge, the only available Java framework with elaborate support for local search metaheuristics is FOM (Parejo et al., 2003) which unfortunately suffers from issues such as limited code transparency and has not been updated to use the latest Java technologies.

The considerations above led to the development of JAMES: a Java 8 framework for discrete optimization using local search metaheuristics. The framework mainly targets single-objective optimization but has extensions for multi-objective optimization. As Java is one of the most used programming languages, JAMES is a valuable addition

to the currently available tools. It is desirable that such framework is transparent, flexible, well-documented, easy to use, and preferably open source (distributed under a permissive license) and hosted on a generally accessible code sharing platform such as GitHub. Java has the additional advantage of being easily portable across different systems (Windows, Unix). JAMES includes a wide range of well-known local searches such as (stochastic) hill-climbing, tabu search (Glover and Taillard, 1993), variable neighbourhood search (Hansen et al., 2010) and parallel tempering (Earl and Deem, 2005; Thachuk et al., 2009). Releases are deployed to the Maven Central Repository so that JAMES can easily be included as a dependency in other Java applications. The project is licensed under the Apache License v2.0. More information and extensive documentation can be found at `http://www.jamesframework.org`.

The Central Repository is the largest curated Java repository and provides a huge amount of open-source packages. It is the default for several build systems such as Apache Maven and SBT, and can easily be used from other tools including Apache Ant and Gradle. Deploying your project to Central is free and ensures that other people can use it with mostly zero configuration. You can browse the repository at `https://goo.gl/sl0A5`.

The Apache License is a permissive open-source license that is used for all Apache projects and many others in the Central Repository. It pretty much allows to use the software and code for any purpose as long as the required notices are included. More info at `https://goo.gl/iqy0SE`.

First, section 3.2 describes the high-level architecture of the JAMES framework (v1.1). Next, sections 3.3 and 3.4 demonstrate how to define and solve a simple core subset selection problem using one of the available local searches. Section 3.5 explains how a new algorithm can be added to the framework, based on two examples. In section 3.6, we compare several algorithms for the defined core selection problem, using the provided analysis workflow. Next, section 3.7 highlights key differences with existing Java frameworks and compares the performance of several frameworks through computational experiments. In section 3.8 we assess the applicability of JAMES to the well-known and extensively studied travelling salesman problem (TSP). In particular, we investigate whether good approximations can be obtained with a simple implementation in JAMES, for moderately large instances from the travelling salesman problem library (TSPLIB; Reinelt, 1991). This also demonstrates that the framework is not limited to subset selection. Finally, we formulate our conclusions in section 3.9.

3.2 ARCHITECTURE OF JAMES

Figure 3.1 shows the high-level architecture of JAMES. Problem specification and search application are strongly separated so that existing algorithms can easily be applied to obtain solutions for newly implemented problems. Each problem has a specific solution type and a search creates solutions of this type to solve the problem. The search communicates with the problem to obtain random solutions (e. g. used as the default initial solution of a local search) and to evaluate and validate constructed solutions. A generic, flexible problem implementation is provided, which is composed of

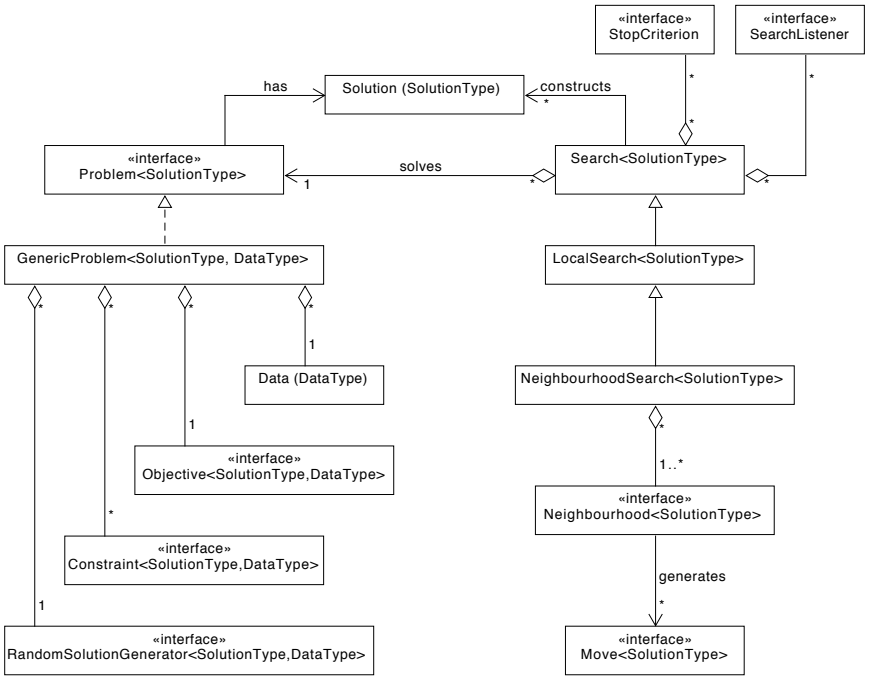


Figure 3.1: High-level architecture of JAMES. Components for problem specification (left) are strongly separated from those related to search application (right). The solution representation lies in between: each problem has a specific solution type and a search constructs solutions of this type to solve the problem.

data, an objective (for evaluation), possibly some constraints (for validation) and a random solution generator.

The optimization algorithms are organized hierarchically. The top-level search definition handles general behaviour such as tracking the best solution found so far and termination (stop criteria). It also informs any listeners when certain events have occurred, e. g. when a new best solution has been found. A local search adds the concept of a current (and initial) solution which is modified in an attempt to improve it, meaning that the search moves towards an optimum along a certain trajectory. The latter is usually performed by repeatedly sampling moves from one or more neighbourhoods that slightly change, and hopefully improve, the current solution. Such algorithms belong to the class of neighbourhood searches. The applied neighbourhoods should be compatible with the solution type of the problem being solved and are used to adjust the search strategy to a specific application.

The lifecycle of a search is depicted in figure 3.2. Each search has a dedicated stop criterion checker, which is activated upon starting the search. This checker runs in a separate thread shared by the stop criterion checkers of all active searches and periodically checks the

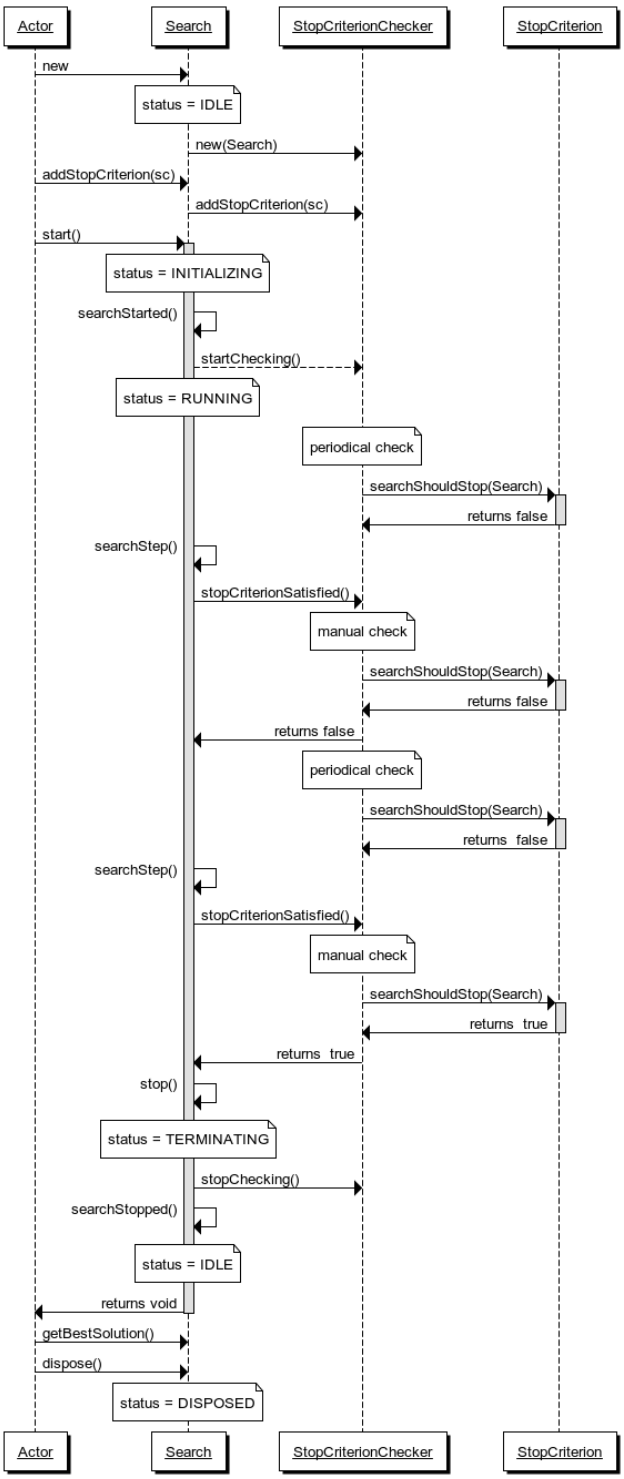


Figure 3.2: Lifecycle of a search terminated by a stop criterion. Upon starting the search, subsequent steps are executed until a stop criterion is satisfied. Stop criteria are checked periodically as well as after each completed search step. When the search has terminated, the best solution can be retrieved after which the search should be disposed so that all resources are properly released. (image made using online *web sequence diagrams*: https://www.websequencediagrams.com)

specified stop conditions (every second, by default). The search itself runs in the thread where it was started and keeps executing search steps until it is requested to stop or terminates internally. After each step, the stop criteria are also checked. When a search has terminated, the best found solution can be retrieved. If desired, an idle search can also be restarted in which case it continues from where it had arrived; in particular, a local search retains its current solution across subsequent runs. Eventually, a search should always be disposed so that all resources are properly released.

The implementation of a search step defines the search strategy. For example, a basic stochastic hill-climber applies a (usually very large) number of steps, in which it attempts to improve the current solution by applying random moves taken from the adopted neighbourhood. Other searches may perform fewer, computationally more intensive steps, or even just one single step. For example, JAMES provides a basic parallel search that consists of a single step in which any possibly heterogeneous collection of searches is executed concurrently, in separate threads. At any time, the best solution found by any of the included searches is returned, and the main search stops when all activated subsearches have completed, where termination request are forwarded to each of these subsearches.

JAMES consists of three modules (*core*, *extensions*, *examples*). The *core* module contains all high-level components shown in figure 3.1, providing the necessary interfaces and (abstract) classes, as well as many algorithm implementations. It also includes specific components for subset selection, such as a predefined solution type, extended problem definition, and several subset neighbourhoods. Similar components can easily be added for other types of problems, and distributed through the *extensions* module when needed—the current version includes additional components for permutation problems. The extensions also provide an automated analysis workflow that can for example be used to compare algorithm performance, to assess the influence of search parameters or to analyse different datasets, problem variants, etc. as well as a weighted index to deal with multi-objective optimization problems, and other utilities. The *examples* module bundles a wide range of examples as described on the website. The next two sections demonstrate how to implement and solve a basic fixed-size subset selection problem. See `http://www.jamesframework.org` for more examples, which also address other types of problems such as the well-known travelling salesman problem (TSP).

Listing 3.1: Data for the core selection problem is provided by implementing the `IntegerIdentifiedData` interface. The data wraps a distance matrix, where item IDs correspond to the row and column indices in the matrix.

```
1  public class CoreSubsetData
2      implements IntegerIdentifiedData {
3
4      private double[][] dist;
5      private Set<Integer> ids;
6
7      public CoreSubsetData(double[][] dist){
8          this.dist = dist;
9          ids = new HashSet<>();
10         for(int id = 0; id < dist.length; id++){
11             ids.add(id);
12         }
13     }
14
15     public double getDistance(int id1, int id2){
16         return dist[id1][id2];
17     }
18
19     public Set<Integer> getIDs(){
20         return ids;
21     }
22
23 }
```

3.3 PROBLEM SPECIFICATION

This section describes the implementation of a simple core subset selection problem. Given a collection of plants and a distance matrix that describes the dissimilarity of each pair, we want to construct a diverse fixed-size subset, with maximum average pairwise distance between selected plants. For such a selection problem, the predefined components can be used, given that a unique integer identifier is assigned to each item. This allows to solve any selection problem by constructing a subset of these IDs.

Core subset selection problems are treated in detail in chapter 4. For practical purposes, there are more appropriate measures to evaluate diversity than average pairwise distance. The simple implementation discussed here merely serves as an example.

A solution type `SubsetSolution` is provided, which tracks the IDs of the selected and unselected items. The corresponding high-level `SubsetProblem` extends `GenericProblem`, fixing the solution type to `SubsetSolution` and specifying a default built-in random subset solution generator. The data class needs to implement the `IntegerIdentifiedData` interface, which defines a single method `getIDs()` used to obtain the set of all assigned IDs. Listing 3.1 shows the implementation of a custom `CoreSubsetData` class that

Listing 3.2: The core selection objective is defined by implementing the `Objective` interface with solution type `SubsetSolution` (predefined) and data type `CoreSubsetData` (custom). A given subset is evaluated by computing the average pairwise distance between all selected items, which is to be maximized.

```
1  public class CoreSubsetObjective
2      implements Objective<SubsetSolution, CoreSubsetData>{
3
4      public Evaluation evaluate(SubsetSolution solution,
5                                 CoreSubsetData data){
6          int n = solution.getNumSelectedIDs();
7          int num = n*(n-1)/2;
8          double sum = 0.0;
9          int[] sel = new int[n];
10         int t = 0;
11         for (int id : solution.getSelectedIDs()) {
12             sel[t++] = id;
13         }
14         for(int i = 0; i < n; i++){
15             for(int j = i+1; j < n; j++){
16                 sum += data.getDistance(sel[i], sel[j]);
17             }
18         }
19         return SimpleEvaluation.WITH_VALUE(sum/num);
20     }
21
22     public boolean isMinimizing() {
23         return false;
24     }
25
26 }
```

wraps a distance matrix, where the item IDs correspond to the row and column indices in this matrix.

The objective is defined by implementing the `Objective` interface and specifying the solution and data type. As dictated by the generic subset problem definition, the solution type is fixed to `SubsetSolution`. For this specific example, we set the data type to `CoreSubsetData`. The objective is responsible for evaluating a given solution, using the data, and informs the search whether these evaluations are to be maximized or minimized. Listing 3.2 shows an implementation of the core selection objective, which evaluates a subset by computing the average pairwise distance between the selected items (lines 4 to 20). As this value is to be maximized, we return `false` in `isMinimizing()` so that the applied search knows that solutions with higher values are preferred (lines 22 to 24).

In addition to the required full evaluation one may optionally also specify an efficient delta evaluation, taking into account that a typical local search evaluates sequences of similar, neighbouring solutions. Hence, we need not evaluate each visited solution from scratch when we know that it has been produced by slightly modifying a similar, already evaluated solution. Instead, the current solution's evaluation can be updated in correspondence with the applied modification. The JAMES framework takes full advantage of such efficient delta evaluations which can significantly speed up the execution with little additional implementation cost (see section 3.7). If no delta evaluation is provided by the user, moves are automatically evaluated by applying them to the current solution, followed by a full evaluation after which the move is undone. This allows for rapid prototyping of a problem specification. An efficient delta evaluation can easily be added later, if and when needed, for example to deal with large problem instances.

Listing 3.3 extends the core selection objective with a delta evaluation for moves of type `SwapMove` (one of the predefined move types for subset solutions; see section 3.4). This delta evaluation runs in $\Theta(n)$ time while the full evaluation has a time complexity of $\Theta(n^2)$, where n is the selection size. Both evaluation methods return a `SimpleEvaluation` that wraps a double value. More complicated evaluation types can be returned as well, for example storing relevant metadata used to compute delta evaluations; examples are found at the website.

Now that the data and objective have been defined, they can be combined in a `SubsetProblem` (listing 3.4). The desired subset size is specified as well (line 10). There are no additional constraints for the considered core selection problem, i. e. all possible subsets of the desired size are valid solutions. Also, since the high-level subset problem definition is already capable of generating random subset solutions, this does not need to be addressed here.

3.4 SEARCH APPLICATION

Once a problem has been defined, the various available optimization strategies can be explored to obtain high-quality solutions. This section demonstrates how to apply a basic stochastic hill-climber (random descent; see section 2.2.4) to the core selection problem as defined in section 3.3. This method starts from a random solution and iteratively applies randomly chosen moves, from a given neighbourhood, to modify the current solution. A move is accepted if and only if it improves the current solution; else, a different move is tried

Listing 3.3: An efficient delta evaluation for specific move types can easily be added to an objective. Here, the objective is designed to be used in combination with a neighbourhood that generates swap moves.

```
1  public class CoreSubsetObjective
2      implements Objective<SubsetSolution, CoreSubsetData>{
3
4      // ... (same as before)
5
6      public Evaluation evaluate(Move move,
7                                 SubsetSolution curSolution,
8                                 Evaluation curEvaluation,
9                                 CoreSubsetData data) {
10
11         SwapMove swapMove = (SwapMove) move;
12
13         // get current evaluation
14         double curEval = curEvaluation.getValue();
15         // undo average to get sum of distances
16         int numSel = curSolution.getNumSelectedIDs();
17         int numDist = numSel * (numSel - 1) / 2;
18         double sumDist = curEval * numDist;
19
20         // retrieve added and removed ID from move
21         int add = swapMove.getAddedID();
22         int del = swapMove.getDeletedID();
23
24         // update distance sum
25         sumDist += curSolution.getSelectedIDs().stream()
26             .mapToDouble(
27                 id -> data.getDistance(add, id)
28                     - data.getDistance(del, id)
29             ).sum();
30         // correct
31         sumDist -= data.getDistance(add, del);
32
33         // return updated evaluation
34         double newEval = sumDist / numDist;
35         return SimpleEvaluation.WITH_VALUE(newEval);
36
37     }
38
39 }
```

Listing 3.4: The core selection problem is finalized by combining the defined data and objective in a `SubsetProblem` with data type `CoreSubsetData` and specifying the desired subset size.

```
1  // specify distance matrix (e.g. read from file)
2  double[][] dist = ...
3
4  // initialize data
5  CoreSubsetData data = new CoreSubsetData(dist);
6  // create objective
7  CoreSubsetObjective obj = new CoreSubsetObjective();
8
9  // specify desired subset size
10 int size = ...
11 // finalize problem
12 SubsetProblem<CoreSubsetData> problem;
13 problem = new SubsetProblem<>(data, obj, size);
```

(in the next search step). Several predefined subset neighbourhoods are available, that can be used for any selection problem. Here, a `SingleSwapNeighbourhood` is applied, which removes a random item from the selection and replaces it with a random, currently unselected item (listing 3.5). This neighbourhood generates moves of type `SwapMove` for which an efficient delta evaluation has been provided in the objective (listing 3.3).

A variety of stop criteria can be used to decide when the search should terminate, such as a runtime or step count limit, or a maximum amount of time or number of steps without finding any improvements. In this example, a runtime limit of 30 seconds is set. Calling `search.start()` (line 12) executes the optimization algorithm in the current thread, after which the best found solution and corresponding value can be retrieved. Finally, the search should be disposed so that all resources are properly released (line 19).

For many applications a simple hill-climber may not be powerful enough to find high-quality solutions because it can not escape from local optima. However, for the considered core selection problem it performs very well and there is no need to turn to more advanced methods (see section 3.6). Examples of more complex problems (including knapsack, TSP and maximum clique) which are solved using other techniques such as parallel tempering or variable neighbourhood search, with both predefined and custom neighbourhoods, are provided at the website.

Listing 3.5: A good core subset, in terms of the average pairwise distance objective, is constructed by applying a simple stochastic hill-climber (random descent) with a single-swap neighbourhood.

```
// create neighbourhood
Neighbourhood<SubsetSolution> neigh
    = new SingleSwapNeighbourhood();
// create search to solve problem
RandomDescent<SubsetSolution> search
    = new RandomDescent<>(problem, neigh);
// set 30 second time limit
StopCriterion sc = new MaxRuntime(30, TimeUnit.SECONDS);
search.addStopCriterion(sc);

// execute search
search.start();
// print solution and value
System.out.println("Solution: "
    + search.getBestSolution().getSelectedIDs());
System.out.println("Value: "
    + search.getBestSolutionEvaluation());
// dispose search to release resources
search.dispose();
```

3.5 ADDING NEW ALGORITHMS

To add a new optimization algorithm to the framework it is sufficient to identify the appropriate entry point in the search hierarchy (see figure 3.1) and to implement the single abstract method `searchStep()`. Possible entry points are:

- `Search`: general search that does not require any additional predefined functionality. Stores the problem being solved and tracks the best found solution. Executes the main search loop and manages high-level behaviour such as stop criteria and search listeners. *Examples: random search, basic parallel search, exhaustive search.*
- `LocalSearch`: adds the concept of a current solution and methods to retrieve and update it. Upon starting the search, a random initial solution is generated if none has been set. *Examples: piped local search, LR subset search.*
- `NeighbourhoodSearch`: modifies the current solution by applying moves sampled from one or more neighbourhoods. Methods are provided to validate and evaluate moves, check whether a move yields a valid improvement, get the best move from a collection of generated moves, accept and re-

Listing 3.6: Implementation of a random sampling strategy.

```
1  public class RandomSearch<S extends Solution>
2         extends Search<S> {
3
4      public RandomSearch(Problem<S> problem){
5          super("MyRandomSearch", problem);
6      }
7
8      protected void searchStep() {
9          Random rnd = getRandom();
10         S sol = getProblem().createRandomSolution(rnd);
11         updateBestSolution(sol);
12     }
13
14 }
```

ject moves, etc. Any local search that uses generic neighbourhoods should extend this class; yet, not directly, but through one of the more specific subclasses (see below).

- `SingleNeighbourhoodSearch`: uses a single neighbourhood. *Examples: random descent, steepest descent, tabu search, parallel tempering.*
- `MultiNeighbourhoodSearch`: uses multiple neighbourhoods. *Examples: variable neighbourhood descent, (reduced) variable neighbourhood search.*

The source code of the many provided searches serves as an example for those interested in extending the framework with custom algorithms. The application programming interface (API) that can be consulted at the website includes detailed documentation of all predefined utility methods. Here, two simple example implementations are discussed: a purely random search (top-level) and the random descent local search strategy (single-neighbourhood) as applied in section 3.4.

3.5.1 *Random search*

One very basic search strategy is as follows: in each step, sample an independent random solution and update the best found solution accordingly. Of course, this method does not have much direct practical value but it may, for example, be used to assess the performance of other algorithms compared to random sampling. No additional functionality is required, so this algorithm is implemented by extending the top-level `Search` class (listing 3.6). Searches

are parameterized on the solution type `S` of the problem being solved, which is required to be a subtype of `Solution` (line 1). The constructor takes a problem with solution type `S`, passes it to the super class and specifies a name for the search (lines 4 to 6). The actual search strategy is implemented in `searchStep()` (lines 8 to 12). Each search has a dedicated random generator, that can be retrieved with `getRandom()` and customized using `setRandom(rnd)`. This generator should be used as the source of randomness in all randomized search components such as neighbourhoods and random solution generators. Having a dedicated random number generator per search avoids contention and consequent performance losses caused by sharing a global generator between possibly multiple active searches, that may be executed concurrently, and allows replicability when assigning a custom random generator with a fixed seed to the applied search(es). Here, it is used to create a random solution by calling the appropriate method, as defined by the `Problem` interface, on the problem that is being solved. The constructed solution is then passed to `updateBestSolution(...)`. As specified in the API, calling this method results in validation and evaluation of the given solution after which the best found solution is updated in case the new solution is a valid improvement.

3.5.2 *Random descent*

Instead of random sampling it is usually better to restrict randomness to a certain neighbourhood, in a local search strategy. Listing 3.7 shows an implementation of a simple stochastic hill-climber (random descent; see section 2.2.4). In each step, a random move is sampled from a given neighbourhood. The move is accepted if it yields a valid improvement when applied to the current solution. This algorithm is implemented as a `SingleNeighbourhoodSearch`, again parameterized on the solution type `S` of the problem (line 1). The constructor now also takes a generic neighbourhood which is passed to the super class (lines 4 to 7). The neighbourhood needs to be *compatible* with the solution type `S` of the search. More precisely, it is allowed to be defined for any super type of `S`. This requirement is sufficient to ensure that the neighbourhood is able to generate moves applicable to solutions of type `S` and promotes flexibility.

The actual search strategy is again implemented in the method `searchStep()` (lines 9 to 24). First, the neighbourhood is retrieved and used to sample a random move for the current solution, using the search's dedicated random generator (line 10). The precise move type is unknown, but guaranteed to be compatible with the solution type `S` (i. e. defined for a super type of `S`) as explained as a

Listing 3.7: Implementation of a random descent stochastic hill-climbing strategy (single-neighbourhood local search).

```
1  public class RandomDescent<S extends Solution>
2         extends SingleNeighbourhoodSearch<S> {
3
4      public RandomDescent(Problem<S> problem,
5                           Neighbourhood<? super S> neigh){
6          super("MyRandomDescent", problem, neigh);
7      }
8
9      protected void searchStep() {
10         Move<? super S> move = getNeighbourhood()
11                                .getRandomMove(
12                                    getCurrentSolution(),
13                                    getRandom()
14                                );
15         if(move != null){
16             if(isImprovement(move)){
17                 accept(move);
18             } else {
19                 reject(move);
20             }
21         } else {
22             stop();
23         }
24     }
25
26 }
```

requirement before. In the unusual case that no move could be produced, the search stops (line 22). Otherwise, it is assessed whether the move yields a valid improvement (line 16). If so, it is accepted (line 17); else, it is rejected (line 19). If the move is accepted, it is applied to the current solution and the best solution is updated accordingly. If the move is rejected, no specific action is to be taken but calling `reject(move)` ensures that the necessary search statistics are updated (for example, the number of accepted/rejected moves is tracked during execution). A detailed description of each predefined method used in this example, and many other methods, is provided in the API documentation available from the website.

3.6 AUTOMATED ANALYSIS WORKFLOW

The extensions module of JAMES includes an automated analysis workflow that can be used to compare algorithm performance, fine-tune parameter values, etc. In contrast to many other metaheuristics

frameworks JAMES itself deliberately does not provide analysis features such as statistical hypothesis testing. Experiments are set up and executed in Java and an R package is used to process and visualize the results. We believe that for analysis of the results it is better to "pass the buck"—using the terminology from Carey and Carlson (2002)—to a specialized software environment. A sound statistical analysis requires a careful setup and providing "push-the-button" statistics within a metaheuristics framework may seduce the user to apply tests or interpret results incorrectly. By shifting statistical analysis to R users have full control and can easily cooperate with statisticians who are used to work with this software. In addition, R provides a wide range of visualization tools.

This section demonstrates the analysis workflow by comparing the performance of a simple stochastic hill-climber (random descent) and the more advanced parallel tempering algorithm—both described in section 2.2.4—when applied to the core selection problem as defined in section 3.3. We have compared the performance for two datasets: a coconut collection with 1014 entries (Odong et al., 2011, 2013) and a pea collection containing 1283 items (Smỳkal et al., 2008). Both distance matrices have been computed based on genetic marker data (see De Beukelaer et al., 2017; supplementary datasets 1 and 2).

Because the considered metaheuristics are randomized, multiple independent runs are performed from which important statistics are inferred such as the average solution quality, variability across runs, and convergence times. This can easily be achieved in JAMES using the analysis workflow (listing 3.8). For each dataset, a distinct problem is added to the analysis, where the data wraps the respective distance matrix (lines 1 to 28). The size of the selected core subset is relative to that of the entire collection (20%). Each problem is assigned a unique ID, here *"dataset-1"* (coconut) and *"dataset-2"* (pea). Both applied algorithms use the predefined single-swap neighbourhood (see section 3.4) and are also assigned a unique ID, here *"Random Descent"* and *"Parallel Tempering"* (lines 31 to 55). When adding a search to the analysis, a factory is given instead of a plain search object, which is used to create an instance of the search given the problem to solve. Both algorithms are executed 10 times (line 58) with a runtime limit of 2 minutes per run (line 35). For each analysed problem and independent search run, a new instance of the search will be created using the provided factory. Note that in the performed analysis, we used the basic objective without delta evaluation (listing 3.2).

Listing 3.8: Example code used to execute multiple independent runs of two algorithms (random descent, parallel tempering) applied to the core selection problem, for two datasets, using the automated analysis workflow from the JAMES extensions module. The size of the selection is relative to that of the entire dataset (20%). Both algorithms use the predefined single-swap neighbourhood and are executed 10 times, for each dataset, with a runtime limit of 2 minutes per run. Results are exported in JSON format.

```
// read dataset files
List<String> datasetFiles = Arrays.asList(
    "coconut/file/path", "pea/file/path"
);
List<CoreSubsetData> datasets = new ArrayList<>();
for(String file : datasetFiles){
    // read distance matrix (implementation omitted)
    double[][] dist = ...
    datasets.add(new CoreSubsetData(dist));
}

Analysis<SubsetSolution> analysis = new Analysis<>();

// add problems
double ratio = 0.2;
CoreSubsetObjective obj = new CoreSubsetObjective();
for(int d = 0; d < datasets.size(); d++){
    CoreSubsetData dataset = datasets.get(d);
    // set size
    int dataSize = dataset.getIDs().size();
    int coreSize = (int) Math.round(ratio * dataSize);
    // create problem
    SubsetProblem<CoreSubsetData> problem
        = new SubsetProblem<>(obj, dataset, coreSize);
    // add to analysis
    String datasetID = "dataset-" + (d+1);
    analysis.addProblem(datasetID, problem);
}

// initialize neighbourhood
Neighbourhood<SubsetSolution> neigh
    = new SingleSwapNeighbourhood();

// set time limit
StopCriterion timeLimit = new MaxRuntime(120, TimeUnit.SECONDS);

// add random descent
analysis.addSearch("Random Descent", problem -> {
    Search<SubsetSolution> rd = new RandomDescent<>(
        problem, neigh
    );
    rd.addStopCriterion(timeLimit);
    return rd;
});
```

```
45 // add parallel tempering
46 analysis.addSearch("Parallel Tempering", problem -> {
47     double minTemp = 1e-8;
48     double maxTemp = 1e-4;
49     int nRep = 10;
50     Search<SubsetSolution> pt = new ParallelTempering<>(
51         problem, neigh, nRep, minTemp, maxTemp
52     );
53     pt.addStopCriterion(timeLimit);
54     return pt;
55 });
56
57 // run analysis
58 analysis.setNumRuns(10);
59 AnalysisResults<SubsetSolution> results = analysis.run();
60 results.writeJSON(
61     "comparison.json", JsonConverter.SUBSET_SOLUTION
62 );
```

By default, a single burn-in run is also executed for each combination of applied algorithm and analysed problem. The results of this run are discarded. Thus, running the example analysis takes about 1.5 hours (2 datasets $\times$ 2 algorithms $\times$ 11 runs $\times$ 2 minutes). Once complete, the results are exported in JSON format (see De Beukelaer et al., 2017; supplementary JSON file) where the specified converter is used to translate the best found solutions to a JSON representation. Here, we use a predefined subset solution converter. If no converter is specified, the produced JSON output will not contain the actual solutions but only their values as well as the history of best solution updates (values and times).

To inspect the results we use the R package `james.analysis` which can directly load the produced JSON file (listing 3.9). The package is available on CRAN and can easily be installed from within R (line 2). After reading the JSON file (line 6) the results can be summarized using the standard R function `summary` (line 7). For each combination of analysed problem and applied search, the summary reports the number of runs, the mean obtained value and corresponding standard deviation, as well as the respective median and interquartile range. It is immediately clear that in this case study, both algorithms are able to construct equally good solutions with low variability across independent runs, for both datasets.

The Comprehensive R Archive Network (CRAN) is more or less for R what the Central Repository is for Java. It is the default location from which packages are installed, and if you want to truly reach the R community with your own package there is no better way than deploying the package to CRAN. Our R package "james.analysis" is available on https://goo.gl/GPbHN0.

Next, some plots are made to gain insight into the convergence of the applied searches (lines 15 to 21; figure 3.3). The random descent algorithm converges after less than one second for both datasets, while parallel tempering converges after 1.5 up to 4 seconds for the coconut and pea dataset, respectively. It is of course expected that parallel tempering is slower than random descent, since it in-

Listing 3.9: The R package `james.analysis` can be used to process and visualize results obtained using the analysis tools from the JAMES extensions module. In this example, a JSON file containing analysis results for the example case study is read, the results are summarized and some plots are made to assess the convergence of the two applied algorithms (random descent and parallel tempering) for the two datasets (see figure 3.3).

```
# install (first time only) and load package
> install.packages("james.analysis")
> library(james.analysis)

# read JSON and print summary
> results <- readJAMES("comparison.json")
> summary(results)
Problem:    Search:              Runs:  Mean value:  St. dev:   Median:      IQR:
---------   ------------------   -----  -----------  --------  --------  --------
dataset-1   Parallel Tempering      10        0.776  1.59e-15     0.776  2.66e-15
dataset-1   Random Descent          10        0.776  2.76e-15     0.776  3.55e-15
dataset-2   Parallel Tempering      10        0.593  1.51e-06     0.593  2.76e-06
dataset-2   Random Descent          10        0.593  6.93e-07     0.593  1.26e-06

# plot convergence curves
> plotConvergence(results, problem = "dataset-1", max.time = 5000)
> plotConvergence(results, problem = "dataset-2", max.time = 5000)

# box plots (convergence times)
> boxplot(results, problem = "dataset-1", type = "time", ylim = c(0, 4500))
> boxplot(results, problem = "dataset-2", type = "time", ylim = c(0, 4500))
```

volves more intensive computations. The convergence curves from figure 3.3 (top row) again confirm that both algorithms obtain the same solution quality. These results suggest that there is no advantage when using parallel tempering instead of a simple stochastic hill-climber to construct core collections with maximum average pairwise distance, as was already observed in previous research (De Beukelaer et al., 2012). Box plots similar to those in figure 3.3 (bottom row) can also be made to assess differences in obtained solution quality. Applicable examples are provided at the website. Documentation of all available functions for data manipulation, extraction and visualization is included in the R package.

3.7 COMPARISON WITH OTHER FRAMEWORKS

Although existing Java metaheuristics frameworks mainly focus on population-based algorithms and especially evolutionary algorithms, they often also provide some support for local searches. Most frameworks include a basic hill-climber and/or simulated annealing. All Java frameworks reviewed by Parejo et al. (2012) except from JCLEC (Ventura et al., 2008) and ECJ (White, 2012) provide

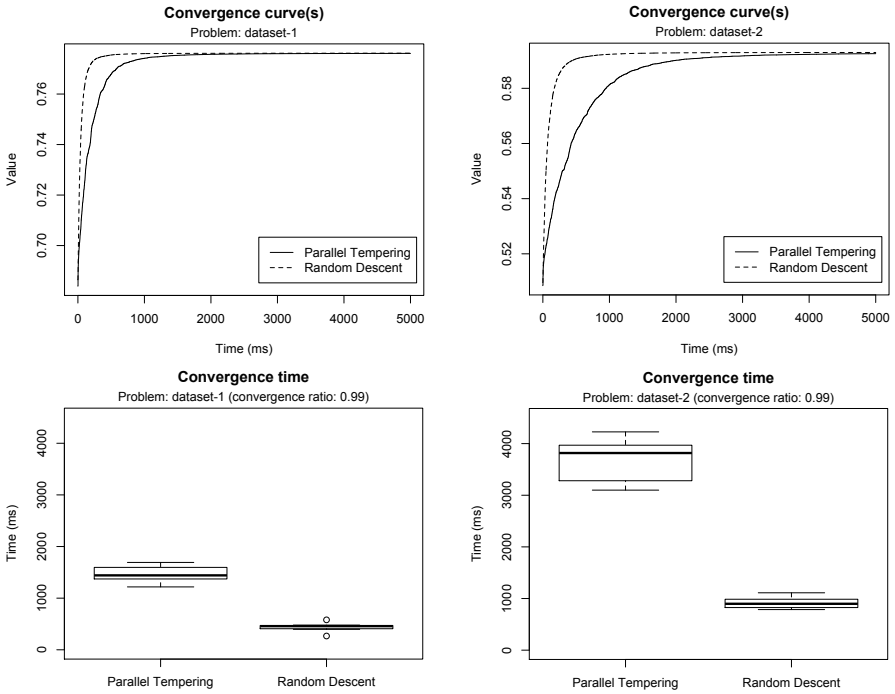


Figure 3.3: Convergence curves (top) and convergence time box plots (bottom) of random descent and parallel tempering applied to sample core collections from the coconut (left) and pea (right) datasets. The size of the core was set to 20% of the full set. Values are reported for 10 independent runs (averaged for the convergence curves). The convergence ratio is set to 0.99 (default) which means that the point in time is reported at which 99% of the progress from initial to final solution has been made.

support for local search algorithms: EvA2 (Kronfeld et al., 2010), Opt4j (Lukasiewycz et al., 2011), OAT (Brownlee, 2007) and FOM (Parejo et al., 2003). Although the review also claims that ECJ (White, 2012) includes a hill-climber we did not find any reference to this feature in the user manual, examples or class documentation. We also excluded OAT from the comparison because its architecture does not allow to plug in custom components like a mutation operator (neighbourhood) and random solution generator without modifying the search class itself, which hinders a fair comparison of the original algorithms as provided by the frameworks. On the other hand, jMetal (Durillo and Nebro, 2011; not covered by Parejo et al., 2012) provides a local search operator which, although intended to be used as part of other algorithms, can be executed separately as well. We thus included the following frameworks for a computational comparison with JAMES: EvA2 (v2.2.0), Opt4j (v3.1.4), jMetal (v5.0) and FOM (v0.5).

We have implemented the simple core selection problem from section 3.3 in each tested framework and assessed the runtime and memory usage of a stochastic hill-climbing algorithm applied to the pea dataset from section 3.6. In EvA2, Opt4j, jMetal and FOM we used a binary solution encoding which indicates for each item whether it is selected (1) or not (0). A solution is evaluated by retrieving the indices of the selected items, followed by calculating the average pairwise distance similar to the JAMES objective from listing 3.2. To create a random neighbour we infer the indices of the selected and unselected items and perform a random swap, similar to the `SingleSwapNeighbourhood` applied in JAMES, which corresponds to two bit flips for a binary encoding. As Opt4j and FOM do not include a stochastic hill-climber we have created one based on the provided simulated annealing implementation by removing the cooling mechanism.

Besides JAMES, only FOM includes the concept of a movement from one solution to another. In FOM it is mainly used to implement certain types of tabu search memories, but we were also able to use this concept to incorporate an efficient delta evaluation by modifying the solution encoding to include the preceding solution and the applied move. For both JAMES and FOM we considered two versions of the implementation—with and without delta evaluation. None of the other frameworks explicitly models moves between solutions, for the obvious reason that a typical population-based algorithm does not benefit from such feature. Therefore, the implementations in these frameworks were restricted to full evaluation of all generated solutions.

For each framework, we assessed the runtime and memory usage when varying either the size of the selection or the number of executed search steps (figure 3.4). Memory usage corresponds to the total sum of allocated memory during the execution of the program and was measured by parsing garbage collection logs. The results clearly show a big difference in memory consumption of the considered frameworks (figure 3.4; top left). The memory usage does not depend on the selection size in any of the frameworks because the employed solution encodings model both the selected as well as unselected items. Therefore, the size of a solution object is determined by the size of the dataset—not of the selected core. EvA2 is the least memory efficient framework, followed by Opt4j, jMetal and FOM, in this order. The latter two consume a very similar amount of memory. Finally, JAMES has a much smaller memory footprint than all other frameworks. Interestingly, the memory consumption of FOM and JAMES does not change when enabling the efficient delta evaluation. JAMES always consumes very little memory, while FOM consistently uses about the same amount of memory as jMetal.

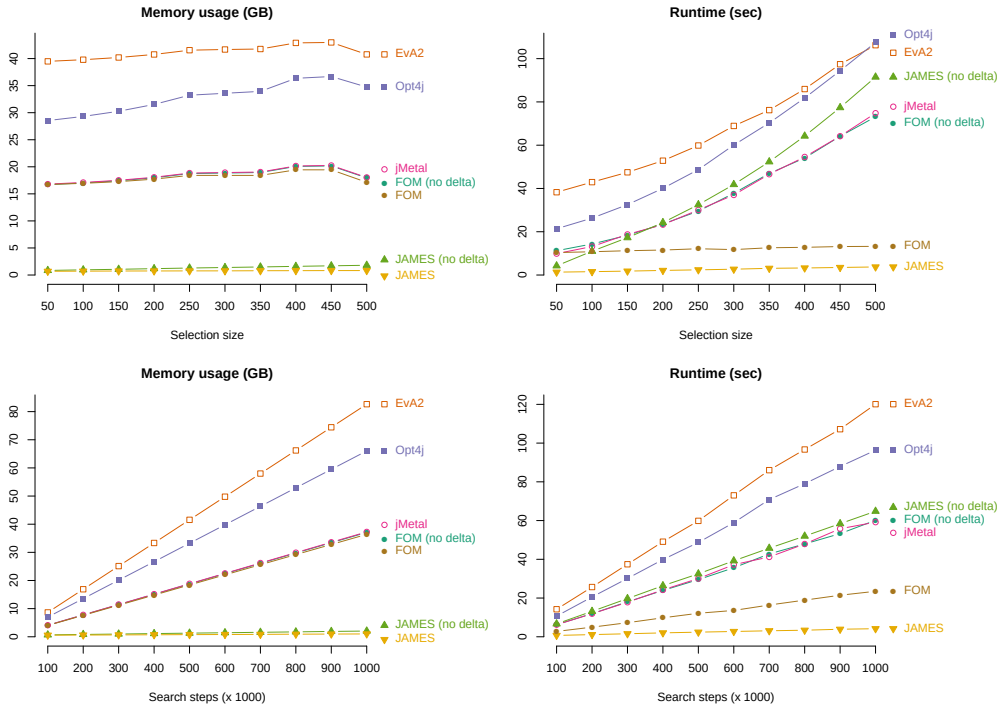


Figure 3.4: Performance of JAMES as compared to that of EvA2, Opt4j, jMetal and FOM when applying a basic stochastic hill-climber to solve the considered core selection problem. Both memory usage (left) and execution time (right) are shown for a varying selection size (top; 500 thousand search steps) or number of executed search steps (bottom; 250 selected items). Averages of 5 independent repeats are reported.

This is because JAMES uses the concept of moves between solutions to avoid excessive copying, even when no efficient delta evaluation is specified, while in FOM the current solution is always copied when requesting a neighbouring solution. In JAMES, neighbourhoods generate moves instead of modified solution copies. If a move is accepted it is applied to the current solution to modify it in place. Copies are only made when a new best solution is found so that it can be stored for later retrieval. On the other hand, FOM generates neighbouring solution objects, which involves copying, and only uses moves as an auxiliary concept that for example allows to specify an efficient delta evaluation and to implement advanced movement-based tabu search memories. This however does not reduce the memory consumption. The large memory footprint of the other frameworks is also mainly attributed to solution copying, which is inherent to population-based algorithms but can and should preferably be avoided in local searches.

The great advantage of specifying an efficient delta evaluation becomes clear when looking at the execution time for varying subset

sizes (figure 3.4; top right). As expected, when delta evaluation is disabled, all frameworks show a quadratic relation between subset size and runtime due to the computation of the average distance between each pair of selected items. EvA2 was the slowest of all evaluated frameworks, followed by Opt4j, which showed to be faster for smaller selection sizes. Both jMetal and FOM are notably faster, with very similar execution times. For small sizes (< 200) JAMES was faster than jMetal and FOM while it was slower for larger selections. This might be caused by the different solution encodings, which both have certain advantages (see below).

Enabling the efficient delta evaluation in JAMES and FOM drastically reduces execution times for both frameworks, with JAMES now being notably faster than FOM. This is because in FOM solutions are still copied which does not only largely increase the memory footprint, as shown before, but also takes time and leads to an increased garbage collection overhead. We conclude that one of the main advantages of JAMES is that it promotes efficient implementations that take full advantage of move-based evaluation.

When increasing the number of search steps (figure 3.4; bottom) both the execution time and memory usage of all frameworks show a linear increase, as expected. Indeed, each step performs exactly the same operations meaning that the time and memory per step do not change during execution. Again, we see that JAMES has a much smaller memory footprint than the other frameworks, and allows to more effectively reduce runtimes by including an efficient delta evaluation as compared to FOM, which is the only other Java framework that explicitly models movements.

To further reduce execution time and memory usage one may design a custom solution encoding tailored specifically to the considered fixed-size core selection problem. For example, we could store two arrays containing the IDs of the selected and unselected items, respectively. Although a binary encoding saves memory and can be efficiently updated, the alternative encoding has the advantage that we can sample a random swap in constant time and that no conversion is needed during evaluation. Such encoding would also improve the performance of JAMES, as the predefined `SubsetSolution` stores IDs in a `Set` to support, for example, variable-size selection and ordered subsets. Since these features are not used here we could easily strip down the solution representation. In JAMES, there is no restriction on the solution encoding and defining a custom solution type is easy and well-documented. Several examples are provided on the website. Other frameworks, including EvA2, Opt4j and jMetal, are often confined to variable-based encodings, which is customary for evolutionary algorithms

but may be unnecessarily restrictive in the context of a local search. Also, not all frameworks allow easy definition of custom or extended solution encodings, e. g. because they do not use generics. For example, this is the case in EvA2 and the original jMetal (up to v4.5). However, jMetal is currently undergoing a complete redesign (starting from v5.0) to take full advantage of the latest Java technology. We took the opportunity to experiment with the new jMetal for our experiments and believe that the redesign is a great success from which many users will benefit. The code of our implementation of the core selection problem in the various frameworks is available at `https://github.com/hdbeukel/james-paper-code`.

3.8 TSPLIB BENCHMARK

Although JAMES provides a series of predefined components for subset selection problems, the framework is in no way limited to this type of problem. Other problems can be solved as well, by defining the necessary custom components such as a solution encoding and corresponding neighbourhood. In this section, the performance of a simple implementation in JAMES is assessed for the well-known symmetric travelling salesman problem (TSP). Test instances were selected from the TSPLIB benchmark collection (Reinelt, 1991). From all 111 instances, 14 were discarded:

The travelling salesman problem (TSP) consist of finding the shortest tour that visits a given number of cities. Each city is to be visited exactly once, with the only exception that the roundtrip should end in the city where it started—this is where our travelling salesman lives. There are many variants of TSP. Here we consider a basic symmetric version where the travel distance between cities is always the same in both directions.

1. Instance `linhp318` describes a constrained TSP problem, with a required edge.
2. The three related instances `si175`, `si535` and `si1032` are not correctly formatted according to TSPLIB instructions.
3. All 10 instances with > 5000 cities were also discarded.

These criteria retained a set of 97 TSPLIB instances, for which the optimal tour length is known. We compared the performance of the basic random descent and more advanced parallel tempering algorithms for these instances. Both algorithms use a basic 2-opt move that breaks two edges in the tour and reconnects the respective vertices in the only other valid way. Implementation details, example code and more information about how the temperature range of parallel tempering was chosen, are available at the website (see `http://www.jamesframework.org/examples/tsp`). As TSP has been very extensively studied over multiple decades, many exact and heuristic approaches have been proposed, along with a wide range of algorithmic tricks to speed up computations (Johnson and McGeoch, 1997). The implementation that was used here deliberately does not incorporate any of these tricks. For example, a solution is simply en-

	30 seconds	5 minutes	1 hour
Random descent	11.57 %	11.30 %	11.27 %
Parallel tempering	2.20 %	1.02 %	0.53 %

Table 3.1: Optimality gap of random descent and parallel tempering for the 97 considered TSPLIB instances. All experiments were repeated with three different runtime limits: 30 seconds, 5 minutes and 1 hour. The mean gap of five independent repeats is reported, averaged over all instances.

coded as a list (permutation) of cities in the order in which they are visited, a 2-opt move is performed by reversing a subsequence of this list, and moves are uniformly sampled from all possible 2-opt moves. This basic implementation fits well in the context of using a framework with a strong focus on simplicity.

Table 3.1 shows the average gap to optimality of random descent and parallel tempering with a runtime limit of 30 seconds, 5 minutes and 1 hour. Detailed results per instance are listed in table 3.2. The performance of random descent is very similar for all three runtime limits, with an average gap of about 11.5%. This indicates that random descent has already converged to a local optimum within less than 30 seconds (often a few seconds; results not shown). The optimality gap is significantly reduced to 0.5–2.2% when applying parallel tempering. Here, increasing the runtime limit beyond 30 seconds does yield better approximations. A major advantage of using a framework like JAMES is that once a problem and the corresponding search components, such as neighbourhoods, have been defined, little effort is required to experiment with various optimization strategies that build upon the defined components. In particular, for parallel tempering, the only additional requirement is to set an appropriate temperature range and number of replicas.

Our results suggest that applying parallel tempering with a basic 2-opt neighbourhood may be sufficient to deal with moderately large TSP problems. Since, in practice, new optimization problems are frequently identified, solving those problems comes with a quality-time tradeoff—not only regarding the runtime of the applied algorithms but also for the time and effort needed to implement and fine-tune the search strategy. During this process, simple methods should not be forgotten, as argued by De Corte and Sörensen (2016). Quite often, overly complex strategies are developed for problems where much simpler techniques perform equally well.

JAMES reduces the effort needed by researchers and practitioners to experiment with various well-known local search strategies and

components, facilitating iterative refinement from a simple starting point, such as a basic hill-climber. Depending on the application, such basic approach may already be sufficient for practical needs. If not, the developed search components can easily be plugged into a more advanced search strategy, such as parallel tempering, and/or additional time can be spent to refine these components themselves (e. g. delta evaluation, neighbourhood, solution encoding, etc.). During this process, the benefit of adding additional complexity should be carefully validated, which is also facilitated by using a framework like JAMES with an automated analysis workflow, as demonstrated in section 3.6.

Table 3.2: Relative optimality gap (%) of random descent and parallel tempering for all unconstrained symmetric TSPLIB instances with $\leqslant$ 5000 cities. Reported gaps are averages with standard deviations ($\pm\ldots$) of 5 independent repeats, for a series of different runtime limits (30 seconds, 5 minutes, 1 hour).

	Random descent			Parallel tempering		
Instance	30 sec	5 min	1 hour	30 sec	5 min	1 hour
a280	17.26 ±2.30	15.36 ±1.71	12.89 ±2.22	0.07 ±0.10	0.00 ±0.00	0.00 ±0.00
ali535	14.01 ±2.80	13.14 ±4.05	15.58 ±2.23	4.12 ±0.28	2.90 ±0.40	1.82 ±0.55
att48	5.08 ±1.30	4.66 ±2.38	4.62 ±2.49	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
att532	10.91 ±1.16	12.11 ±1.66	12.20 ±2.06	2.15 ±0.35	0.79 ±0.18	0.35 ±0.11
bayg29	3.33 ±2.04	5.22 ±2.07	6.29 ±1.75	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
bays29	4.18 ±3.15	3.98 ±2.97	5.82 ±3.38	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
berlin52	10.56 ±3.03	12.15 ±1.99	11.43 ±1.43	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
bier127	13.55 ±4.78	10.29 ±1.49	9.99 ±1.25	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
brazil58	4.09 ±1.38	5.34 ±1.81	3.85 ±2.87	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
brg180	25.13 ±5.88	28.72 ±2.66	26.26 ±6.25	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
burma14	2.94 ±3.51	1.40 ±2.49	3.20 ±2.63	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
ch130	9.10 ±1.67	10.90 ±2.52	9.23 ±0.91	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
ch150	13.59 ±2.18	12.71 ±1.18	14.18 ±3.65	0.17 ±0.14	0.00 ±0.00	0.00 ±0.00
d1291	21.09 ±2.68	20.97 ±1.32	22.09 ±1.86	7.50 ±0.32	3.95 ±0.41	2.81 ±0.27
d1655	17.68 ±1.66	17.34 ±1.09	17.69 ±0.26	6.53 ±0.81	3.58 ±0.38	2.15 ±0.23
d198	5.11 ±1.04	5.32 ±0.99	6.27 ±0.83	0.07 ±0.03	0.00 ±0.00	0.00 ±0.00
d2103	23.57 ±1.44	23.33 ±1.52	23.07 ±1.55	11.72 ±0.99	5.73 ±0.64	1.61 ±0.28
d493	11.74 ±1.51	11.14 ±0.97	11.20 ±1.64	1.66 ±0.16	0.71 ±0.14	0.16 ±0.07
d657	13.48 ±1.57	12.52 ±0.79	13.80 ±0.22	2.27 ±0.33	0.68 ±0.04	0.25 ±0.04
dantzig42	10.27 ±2.53	5.64 ±2.62	7.12 ±1.49	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
dsj1000	13.54 ±1.52	13.74 ±2.89	14.65 ±1.73	3.37 ±0.46	1.85 ±0.27	0.75 ±0.06
eil101	9.48 ±3.40	11.57 ±2.02	10.08 ±0.58	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
eil51	10.14 ±5.92	8.03 ±2.21	5.49 ±2.35	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
eil76	10.00 ±3.54	9.48 ±3.64	10.71 ±1.93	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
fl1400	10.39 ±3.12	9.36 ±1.34	10.24 ±2.20	4.58 ±1.29	2.90 ±0.63	3.20 ±0.76
fl1577	21.55 ±1.48	18.34 ±4.34	17.48 ±2.57	6.09 ±1.44	2.06 ±0.41	1.35 ±0.82
fl3795	19.39 ±2.87	17.96 ±1.19	18.69 ±4.41	15.08 ±1.49	6.04 ±0.76	2.90 ±1.48
fl417	9.62 ±1.61	7.78 ±4.28	8.86 ±3.04	0.96 ±0.52	0.69 ±0.33	0.41 ±0.28
fnl4461	13.43 ±0.49	13.61 ±0.21	13.43 ±0.53	15.00 ±0.29	6.10 ±0.36	3.33 ±0.37

fri26	4.44 ±4.09	5.27 ±4.43	3.42 ±3.68	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
gil262	13.04 ±1.61	11.58 ±1.95	12.49 ±0.95	0.29 ±0.13	0.00 ±0.00	0.00 ±0.00
gr120	9.23 ±1.93	7.23 ±0.84	9.00 ±2.98	0.07 ±0.10	0.00 ±0.00	0.00 ±0.00
gr137	9.23 ±1.93	10.66 ±2.61	11.45 ±2.65	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
gr17	1.56 ±1.81	0.72 ±1.13	2.12 ±1.58	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
gr202	9.95 ±0.74	10.88 ±3.25	9.69 ±1.16	0.14 ±0.10	0.00 ±0.00	0.00 ±0.00
gr21	6.87 ±7.95	5.54 ±2.79	3.21 ±4.80	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
gr229	10.91 ±2.38	10.53 ±1.84	11.31 ±2.29	0.45 ±0.14	0.04 ±0.02	0.00 ±0.00
gr24	4.45 ±3.00	4.81 ±3.30	4.09 ±5.15	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
gr431	12.59 ±1.51	9.82 ±1.63	12.13 ±2.40	2.32 ±0.41	1.32 ±0.67	0.43 ±0.23
gr48	5.10 ±3.50	4.84 ±2.25	6.41 ±1.97	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
gr666	15.09 ±1.40	14.20 ±1.43	14.32 ±0.99	3.79 ±0.58	1.99 ±0.55	0.96 ±0.34
gr96	8.34 ±2.18	9.40 ±2.21	8.88 ±2.34	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
hk48	9.17 ±3.11	8.46 ±2.72	5.97 ±2.56	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
kroA100	9.80 ±3.73	11.59 ±1.76	8.85 ±2.83	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
kroA150	12.10 ±1.32	10.66 ±4.33	12.38 ±4.27	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
kroA200	11.43 ±2.58	11.43 ±2.79	11.43 ±1.71	0.20 ±0.07	0.00 ±0.00	0.00 ±0.00
kroB100	8.77 ±2.02	7.92 ±2.95	9.65 ±2.66	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
kroB150	8.07 ±1.60	10.44 ±1.68	10.83 ±3.83	0.01 ±0.01	0.00 ±0.00	0.00 ±0.00
kroB200	12.07 ±2.74	10.84 ±2.88	10.41 ±1.20	0.11 ±0.05	0.00 ±0.00	0.00 ±0.00
kroC100	6.64 ±1.48	9.51 ±0.65	9.45 ±4.13	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
kroD100	9.35 ±3.59	8.81 ±3.76	7.98 ±1.73	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
kroE100	8.22 ±3.60	8.84 ±5.21	12.07 ±3.47	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
lin105	10.38 ±3.24	7.00 ±2.11	10.12 ±2.13	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
lin318	13.85 ±1.19	11.55 ±1.06	11.03 ±3.59	1.34 ±0.44	0.50 ±0.14	0.04 ±0.06
nrw1379	13.15 ±0.69	13.17 ±0.96	12.79 ±0.71	6.14 ±0.85	3.08 ±0.26	1.66 ±0.21
p654	10.06 ±1.76	7.97 ±1.64	10.67 ±3.47	3.31 ±0.85	2.04 ±0.60	0.86 ±0.55
pa561	14.01 ±2.65	14.84 ±1.32	13.81 ±2.15	2.28 ±0.37	0.77 ±0.08	0.28 ±0.07
pcb1173	15.92 ±0.74	15.98 ±1.09	15.37 ±0.94	4.75 ±0.20	2.15 ±0.09	0.91 ±0.07
pcb3038	15.37 ±0.56	15.79 ±0.67	15.78 ±0.76	11.59 ±0.64	5.23 ±0.50	2.99 ±0.08
pcb442	14.96 ±1.34	15.05 ±1.42	13.85 ±2.01	1.30 ±0.41	0.25 ±0.11	0.02 ±0.01
pr1002	13.72 ±1.32	14.04 ±1.52	14.55 ±1.09	3.84 ±0.43	1.81 ±0.09	0.79 ±0.14
pr107	6.30 ±2.43	7.90 ±2.97	10.80 ±2.31	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
pr124	5.49 ±4.12	7.47 ±6.82	3.99 ±1.50	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
pr136	10.14 ±2.32	12.07 ±1.83	10.24 ±2.66	0.23 ±0.08	0.01 ±0.01	0.00 ±0.00
pr144	11.23 ±3.57	4.83 ±4.26	8.87 ±5.59	0.06 ±0.09	0.00 ±0.00	0.00 ±0.00
pr152	6.51 ±2.47	5.12 ±0.83	5.45 ±2.62	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
pr226	7.83 ±3.98	7.46 ±4.80	8.36 ±3.59	0.41 ±0.42	0.17 ±0.11	0.06 ±0.13
pr2392	16.11 ±1.19	16.53 ±0.60	17.49 ±1.80	9.37 ±1.27	4.37 ±0.41	2.58 ±0.51
pr264	15.12 ±3.66	15.03 ±3.55	14.21 ±1.42	0.16 ±0.26	0.00 ±0.00	0.00 ±0.00
pr299	15.46 ±1.28	13.13 ±1.61	12.28 ±1.47	0.36 ±0.17	0.01 ±0.01	0.00 ±0.00
pr439	15.50 ±1.72	13.80 ±1.60	13.83 ±1.99	2.35 ±0.52	1.11 ±0.34	0.84 ±0.52
pr76	6.31 ±1.81	7.86 ±2.17	7.19 ±2.55	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
rat195	12.94 ±0.53	13.05 ±2.31	13.38 ±0.86	0.47 ±0.12	0.09 ±0.04	0.00 ±0.00
rat575	11.79 ±1.53	12.22 ±1.00	13.68 ±1.28	2.76 ±0.47	1.29 ±0.13	0.38 ±0.09
rat783	13.86 ±0.63	14.07 ±1.16	13.35 ±1.05	3.25 ±0.24	1.54 ±0.10	0.57 ±0.10
rat99	13.25 ±4.79	13.06 ±2.36	9.64 ±3.71	0.02 ±0.04	0.00 ±0.00	0.00 ±0.00
rd100	12.97 ±3.47	9.90 ±3.25	11.65 ±2.86	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
rd400	13.01 ±1.48	11.97 ±2.08	12.56 ±2.27	1.28 ±0.25	0.34 ±0.04	0.05 ±0.02
rl1304	18.53 ±1.73	16.16 ±0.77	18.04 ±2.55	7.04 ±1.01	3.95 ±0.51	2.46 ±0.22
rl1323	18.10 ±1.33	16.88 ±2.71	16.25 ±2.03	6.08 ±0.84	3.03 ±0.42	1.41 ±0.30
rl1889	16.64 ±2.52	18.01 ±1.09	16.52 ±1.29	8.83 ±1.16	4.91 ±0.62	2.73 ±0.30

st70	9.54 ±4.03	9.51 ±2.64	7.50 ±2.64	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
swiss42	8.01 ±3.16	11.04 ±3.24	6.79 ±3.29	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
ts225	8.21 ±1.35	9.42 ±1.60	6.41 ±2.71	0.09 ±0.03	0.00 ±0.00	0.00 ±0.00
tsp225	12.22 ±1.33	13.73 ±1.40	12.92 ±1.50	0.94 ±0.19	0.13 ±0.17	0.00 ±0.00
u1060	13.15 ±1.15	14.49 ±1.21	13.43 ±1.06	3.26 ±0.09	1.51 ±0.34	0.66 ±0.13
u1432	15.64 ±0.71	16.59 ±0.94	15.96 ±1.27	5.03 ±0.63	2.76 ±0.53	1.71 ±0.31
u159	12.57 ±0.82	11.18 ±4.45	13.20 ±3.03	0.08 ±0.19	0.00 ±0.00	0.00 ±0.00
u1817	22.51 ±1.35	21.37 ±1.52	20.62 ±1.33	7.92 ±1.05	3.26 ±0.30	1.60 ±0.15
u2152	21.82 ±0.98	21.46 ±0.91	20.02 ±1.49	8.80 ±0.81	3.80 ±0.33	1.93 ±0.34
u2319	9.30 ±0.25	9.09 ±0.43	9.32 ±0.37	3.41 ±0.36	1.40 ±0.12	0.91 ±0.06
u574	13.88 ±1.13	13.03 ±0.52	13.37 ±0.74	2.23 ±0.18	0.83 ±0.18	0.13 ±0.02
u724	13.80 ±1.83	13.60 ±1.23	13.81 ±0.88	2.84 ±0.28	1.14 ±0.05	0.40 ±0.04
ulysses16	0.48 ±0.68	1.83 ±1.57	0.44 ±0.58	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
ulysses22	2.73 ±1.73	2.20 ±1.30	2.02 ±1.84	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
vm1084	14.30 ±1.73	14.01 ±1.38	13.43 ±1.45	5.09 ±0.28	2.60 ±0.26	1.07 ±0.11
vm1748	14.63 ±1.59	15.42 ±0.74	14.21 ±1.04	7.38 ±0.77	3.56 ±0.44	2.05 ±0.09

3.9 CONCLUSIONS

In this chapter we described JAMES (v1.1): an object-oriented Java framework for discrete optimization using local search metaheuristics. By clearly separating problem specification from search application the provided algorithms can easily be used to solve newly defined problems, as was demonstrated for a simple fixed-size core selection problem. We showed how new algorithms can be added, and how to compare the performance of several algorithms using the automated analysis workflow included in the extensions. A comparison with other Java metaheuristics frameworks, that mainly focus on population-based algorithms but have some support for local search techniques, revealed that JAMES has a much lower memory pressure because it avoids excessive solution copies by explicitly modelling moves and maximally exploiting this concept. Avoiding copies also reduces execution time, which can be significantly further improved through an efficient move-based delta evaluation mechanism. Although JAMES includes predefined components for subset selection, the framework is in no way limited to selection problems. To demonstrate that other problems can be solved as well, we applied JAMES to the well-known travelling salesman problem (TSP). We showed that a simple implementation was able to find good approximations for moderately large TSP instances from the TSPLIB benchmark.

In all, the JAMES framework is a valuable addition to the currently available Java metaheuristics optimization tools, which mainly focus on population-based algorithms, and its efficiency is achieved by specifically focusing on local searches. Future work includes

the addition of new algorithms and, when needed, components to model other common optimization problems. The latter will be distributed as part of the extensions module. In chapters 4 and 5 we use JAMES to solve several selection problems, with applications in plant breeding and genetics in general.

IMPLEMENTATION AND HARDWARE

Experiments were implemented in Java 8 and executed on a Linux computing server with two 2.6 GHz 8-core Intel E5-2670 (Sandy Bridge) CPUs and 32 GB RAM. Results were analysed in R, version 3.2.1 (R Core Team, 2015).

Part III

PLANT BREEDING PROBLEMS

In the main part of this thesis we address three optimization problems related to marker-assisted plant breeding: core subset selection, long-term genomic selection strategies, and marker-assisted gene pyramiding. Each of these applications uses genetic marker data to make better decisions in practical plant breeding schemes.

4

MULTI-PURPOSE CORE SUBSET SELECTION

SUMMARY

This chapter describes Core Hunter 3: a flexible tool to sample core collections from genetic resources. Such cores represent the diversity of the full collection, with minimum redundancy, and allow to effectively utilize large resources. Many methods and measures have been proposed to sample and evaluate core collections. Core Hunter uses local search algorithms to optimize one or more criteria, depending on the purpose of the core. Distance-based evaluations are often used because they are easy to interpret and can be computed for both genetic marker data and phenotypic traits. Core Hunter 2 (CH2) maximized average and minimum distance but it was later suggested to instead maximize average entry-to-nearest-entry (E-NE) distance to obtain diverse cores, or to minimize average accession-to-nearest-entry (A-NE) distance to maximally represent all individual accessions from the entire collection. Core Hunter 3 (CH3) was designed to include these improved measures.

We show that the E-NE criterion can be effectively optimized with a simple stochastic hill-climbing algorithm. Still, the diversity of the core is slightly further increased, and variability across independent samples further reduced, by using the parallel tempering algorithm. A more complex algorithm like the mixed replica search used by CH2 is not needed to optimize this measure. Core Hunter 3 yields higher E-NE values than CH2 while still ensuring a high minimum distance, and is faster for large datasets. A comparison with two existing methods revealed that CH3 can sample equally representative cores as GDOpt, which was specifically designed for this purpose, and is able to construct cores that are simultaneously more diverse, and either are more representative or have higher allelic richness, than those obtained by SimEli.

In all, Core Hunter 3 is a flexible, fast and very broadly applicable core subset selection tool that samples multi-purpose cores based on genetic marker data or phenotypic traits. It combines and outperforms the strengths of other methods and can easily be extended with new evaluation measures without the need to alter the underlying optimization algorithms. For more information about the open-source Core Hunter 3 project, visit http://www.corehunter.org.

4.1 INTRODUCTION

In an effort to preserve agricultural biodiversity, seeds of major crops are stored in so-called gene or seed banks. These genetic resources do not only include modern cultivars but also many historical landraces and wild relatives. One of the most famous gene banks is probably the Svalbard Global Seed Vault in Norway, which stores backups of seeds held in gene banks worldwide, as a protection against potential losses in case of, for example, a large-scale crisis at one of these other locations. The Svalbard vault is built inside a sandstone mountain on Spitsbergen Island and highly secured. Recently, the initiative proved useful when, due to the Syrian Civil War, a local gene bank was forced to move from Aleppo to Beirut. Because of the difficulties they encountered when transferring their original collections, the Svalbard vault authorized the first backup seed withdrawal in its history.

Most of the diversity found in cultivated plant species is not directly used for agricultural purposes. Yet, it is very important to maintain this diversity so that plant breeders and researchers can keep looking for novel traits in conserved germplasm. Over time, the collections stored in gene banks have grown enormously, which prohibits the effective characterization and utilization of the full collections. Therefore, Frankel (1984) proposed to sample smaller so-called *core collections* which represent the diversity of the entire collection with minimum redundancy. These cores offer an efficient way to effectively characterize and utilize large genetic resources for future crop improvement.

A variety of measures have been used to evaluate core collections based on genetic marker data or phenotypic traits, including pairwise distances and allelic diversity. As argued by Odong et al. (2013) the choice of the most appropriate evaluation measure depends on the purpose of the core collection. For example, breeders are often interested in cores in which all accessions are sufficiently different from each other, whereas gene bank curators and geneticists in general may prefer to retain rare alleles. Sometimes core collections are sampled based on a combination of both genotypes and phenotypes (Borrayo et al., 2016; Franco et al., 2010; Wang et al., 2006).

Many methods have been proposed to sample high-quality core collections according to the measure(s) of interest. The first methods were stratified sampling techniques that cluster the data and then select several accessions from each cluster using a certain allocation method. Brown (1989) suggested to randomly select either a constant (C) number of accessions per cluster, or a number proportional (P) to the size or logarithm (L) of the size of the cluster, and argued that the L-method is preferred. Franco et al. (2005) later showed that more diverse cores are obtained when the number of included accessions is proportional to the within-cluster diversity.

Another allocation method, the M-method, maximizes the probability to retain all observed alleles in order to construct cores with high allelic richness (Schoen and Brown, 1993). This idea led to the development of the MSTRAT software, which implements a generalized M-method that directly samples from the entire collection to maximize allelic richness with a simple hill-climbing algorithm (Gouesnard et al., 2001). Other heuristics work by repeatedly removing one of the two most similar accessions from the collection until the desired core size is obtained, either randomly (least distance stepwise sampling; Wang et al., 2007), or using a specific elimination criterion maximizing the distance to the remaining accessions

or expected heterozygosity of the reduced collection (SimEli; Krishnan et al., 2014). Odong et al. (2011) designed the genetic distance optimization strategy (GDOpt) to construct highly representative cores, in which each accession from the entire collection is represented by a similar core entry. GDOpt partitions the data around a number of identified medoids which are then selected as the core entries. Methods for variable-size core sampling have also been developed. PowerCore minimizes the size of the core while covering all observed marker alleles and/or trait values (Kim et al., 2007). The genetic distance sampling strategy constructs cores with a given minimum distance between selected accessions by repeatedly including a random accession and removing all others within a certain sampling radius (Jansen and Van Hintum, 2007).

A medoid is a representative item in a group of data, whose average dissimilarity to all other items is minimal. In contrast to a centroid, a medoid is required to be an actual member of the dataset.

Core Hunter was developed to meet the variety of criteria used to evaluate core collections for different purposes, and provides multiple objectives that are optimized using flexible local search algorithms (Thachuk et al., 2009). Core Hunter allows to construct core collections for specific applications, or to combine multiple objectives to bring the different perspectives closer together, for example by simultaneously maximizing pairwise genetic distance and allelic richness. Although Core Hunter is mainly focused at fixed-size core selection, version 1 and 2 allowed to specify a minimum and maximum size and preferred smaller cores with the same value. Core Hunter was shown to outperform other methods including stratified sampling strategies, MSTRAT and PowerCore.

It has been suggested that, to obtain a diverse core, the average distance between its entries should be maximized (Franco et al., 2005; Thachuk et al., 2009). However, a high average entry-to-entry distance does not guarantee that selected accessions are sufficiently different and it is known that maximizing this criterion overrepresents extreme values (De Beukelaer et al., 2012; Odong et al., 2013). Core Hunter 2 (CH2) deals with this issue by also maximizing the minimum distance between selected accessions (De Beukelaer et al., 2012). Although average distance and allelic richness can be effectively optimized using simple and fast local search algorithms such as stochastic hill-climbing, a more complex and slower *mixed replica search* (MixRep) was required to maximize minimum distance in the Core Hunter framework.

Another approach to maximize diversity while at the same time avoiding inclusion of too similar accessions at the extremes of the distribution is to maximize the average distance between each entry and the closest other entry in the core (Odong et al., 2013). The SimEli algorithm was shown to outperform Core Hunter 2 in terms of this entry-to-nearest-entry (E-NE) objective. Alternatively, one

may desire to optimally represent the individual accessions instead of the whole range of diversity. In such case, it is recommended to minimize the average distance between each accession in the full collection and the closest core entry. The GDOpt algorithm was specifically developed to minimize this average accession-to-nearest-entry (A-NE) distance and shown to outperform both Core Hunter 2 and SimEli for this purpose (Krishnan et al., 2014; Odong et al., 2011).

We introduce Core Hunter 3 (CH3) which incorporates the improved E-NE and A-NE criteria proposed by Odong et al. (2013), and can sample fixed-size cores based on molecular marker data, phenotypic trait data, a precomputed distance matrix, or a combination of these. The distance matrix can be generated using an appropriate distance measure such as Modified Roger's distance for genotypes (Wright, 1987) or Gower's distance for phenotypes (Gower, 1971). As in previous versions, Core Hunter 3 can also maximize allelic richness. In particular, we assess whether the new distance-based E-NE and A-NE measures can be effectively optimized using fast local search algorithms, and whether maximizing E-NE indirectly also yields a high minimum distance in the constructed core without the need for a more complex algorithm. Furthermore, we assess the ability of Core Hunter 3 to simultaneously maximize E-NE and A-NE, or E-NE and allelic richness, and compare the results to those obtained with Core Hunter 2, GDOpt, and SimEli, for three marker datasets with different allelic composition and varying size, and one phenotypic trait dataset. Core Hunter 3 is available as an R package on CRAN and as an open-source project on GitHub. A prototype graphical user interface is also provided. See `http://www.corehunter.org` for more information.

The R package `corehunter` is available on `https://goo.gl/Dsb311`.

4.2 HISTORY OF CORE HUNTER

The original Core Hunter software was developed by Thachuk et al. (2009) to sample diverse core collections from large genetic resources based on molecular marker data. Core Hunter 1 (CH1) used the parallel tempering algorithm described in section 2.2.4 to maximize average pairwise genetic distance and several allelic diversity indices, such as expected heterozygosity (Berg and Hamrick, 1997) and Shannon's index (Shannon, 2001).

Many of the algorithms initially implemented in Core Hunter 2 finally found their way into the JAMES framework (see chapter 3).

As a master student I worked on Core Hunter 2 (CH2) and experimented with alternative optimization engines, including many local searches and some population-based metaheuristics, as well as a few simple constructive heuristics. We found that the evaluation

measures supported by CH1 could effectively be optimized with a simple stochastic hill-climber (random descent; see section 2.2.4) without the need for a more complex algorithm such as parallel tempering. However, we also discovered that maximizing average pairwise distance tends to overrepresent the extremes of the distribution. As a solution, we suggested to also maximize the minimum distance between any two selected accessions. Unfortunately, maximizing this new measure is a big challenge for a local search, as slightly modifying the current solution in many cases does not change the minimum distance between selected items. Therefore, the search is not efficiently guided towards an optimum and may even just get stuck with its initial random selection. To overcome this limitation, we used a constructive heuristic called *LR* (left-right) search that starts with an empty selection and then iteratively adds r and removes $l < r$ items, maximizing the objective function with each addition or removal, until the desired core size is obtained.

The LR heuristic showed to be able to effectively maximize the minimum distance between selected items, but is unfortunately quite slow for large datasets. Therefore, we eventually created a hybrid *mixed replica search* (MixRep; De Beukelaer et al. 2012) which runs multiple algorithms in parallel—including random descent and LR search—and exchanges solutions between subsearches, inspired by the parallel tempering algorithm. In this way, users interested in maximizing average genetic distance or allelic diversity quickly get high-quality results, while users who desire a high minimum distance will still obtain good core subsets with the same optimization engine, simply by allowing longer execution times.

Odong et al. (2013) later proposed to sample diverse cores by maximizing the average distance between each selected item and the closest other selected item (entry-to-nearest-entry). Alternatively, to maximally represent each individual accession in the full collection, they suggested to minimize the average distance between each accession and the most similar selected one (accession-to-nearest-entry). Core Hunter 3 incorporates these new distance-based measures which, as shown below, can easily be optimized with fast local searches. Maximizing the entry-to-nearest-entry distance also yields a high minimum distance, and thus eliminates the need for the quite complex and slower mixed replica search from CH2.

Furthermore, Core Hunter 3 now also supports phenotypic trait data in addition to genetic marker data, and various formats instead of a single marker data format. Phenotypic traits are analysed using Gower's distance (Gower, 1971), while for marker data users can choose between the Modified Roger's (Wright, 1987) or Cavalli-Sforza and Edwards (Cavalli-Sforza and Edwards, 1967) distances,

or, alternatively, maximize allelic richness. In addition, a precomputed distance matrix can be provided by the user as well. We reimplemented Core Hunter 3 from scratch using the JAMES framework and made the new version easily accessible through an elaborate R package deployed on CRAN.

4.3 MATERIALS AND METHODS

4.3.1 *Datasets*

We used four datasets of varying size and composition to compare the performance of different core sampling algorithms:

1. *Rice data*: 1000 accessions for which 39 phenotypic traits were recorded, including 28 qualitative and 11 quantitative traits. Available from the PowerCore project (Kim et al., 2007) and used before to assess the performance of several other core sampling algorithms, including SimEli (Krishnan et al., 2014).
2. *Coconut data*: 1014 accessions characterized using 30 crop-specific SSR markers. Used in multiple previous core selection studies (Krishnan et al., 2014; Odong et al., 2011, 2013).
3. *Maize data*: 1250 accessions characterized with 1117 SNP markers. Distributed as part of the R package `synbreedData` (Wimmer et al., 2015).
4. *Pea data*: 4428 accessions characterized by 17 RBIP markers (Jing et al., 2010; Smýkal et al., 2011). Previously used to compare the performance of Core Hunter 2 with other core sampling algorithms for datasets with many accessions (De Beukelaer et al., 2012).

Within Core Hunter, all supported types of marker data are converted to allele frequencies (see section 1.2.2), which are used to compute the various objective functions. All cores sampled in the performed experiments comprise 20% of the full collection for the rice, coconut, and maize datasets, and 10% for the large pea dataset.

4.3.2 *Evaluation measures*

Core Hunter 3 includes various evaluation measures that can be selected as optimization objectives, including but not limited to those described below. For an overview of all provided measures, we refer to the website `http://www.corehunter.org`.

Distance measures

We used the Modified Roger's distance (Wright, 1987) to assess the dissimilarity of accessions based on genetic marker data. For phenotypic traits we used Gower's distance (Gower, 1971) which simultaneously takes into account qualitative and quantitative traits. Pairwise distances are aggregated in a certain way to evaluate the diversity or representativeness of the core (Odong et al., 2013):

- Entry-to-nearest-entry (E-NE): average distance between each selected accession and the closest other core entry. This criterion can be maximized to construct highly diverse cores in which all accessions are maximally different.
- Accession-to-nearest-entry (A-NE): mean distance between each accession from the entire collection and the most similar core entry, including itself in case the accession has been selected. Minimizing this criterion yields cores that optimally represent all individual accessions from the full collection.

As shown in figure 4.1 maximizing E-NE tends to select accessions at cluster edges while minimizing A-NE favours accessions near cluster centers. When comparing CH3 with CH2 we also evaluated the minimum distance (DMIN) between selected accessions but this is not an objective that can directly be optimized by CH3, for reasons explained before and further elaborated on in the discussion.

Allelic richness

To evaluate the allelic richness of cores sampled based on genetic marker data we used the average expected heterozygosity (HE) per locus (Berg and Hamrick, 1997), calculated as

$$0 \leqslant \mathrm{HE} = \frac{1}{L} \sum_{l=1}^{L} (1 - \sum_{a=1}^{n_l} \hat{p}_{la}^2) \leqslant 1$$

where L is the number of markers (loci), n_l is the number of observed alleles at the l-th locus, and $\hat{p}_{la}^2$ is the frequency of the a-th allele at the l-th locus in the selected core collection.

Weighted index and normalization

Core Hunter allows to simultaneously optimize k measures by maximizing a weighted index

$$F(c) = \sum_{i=1}^{k} \alpha_i F_i(c)$$

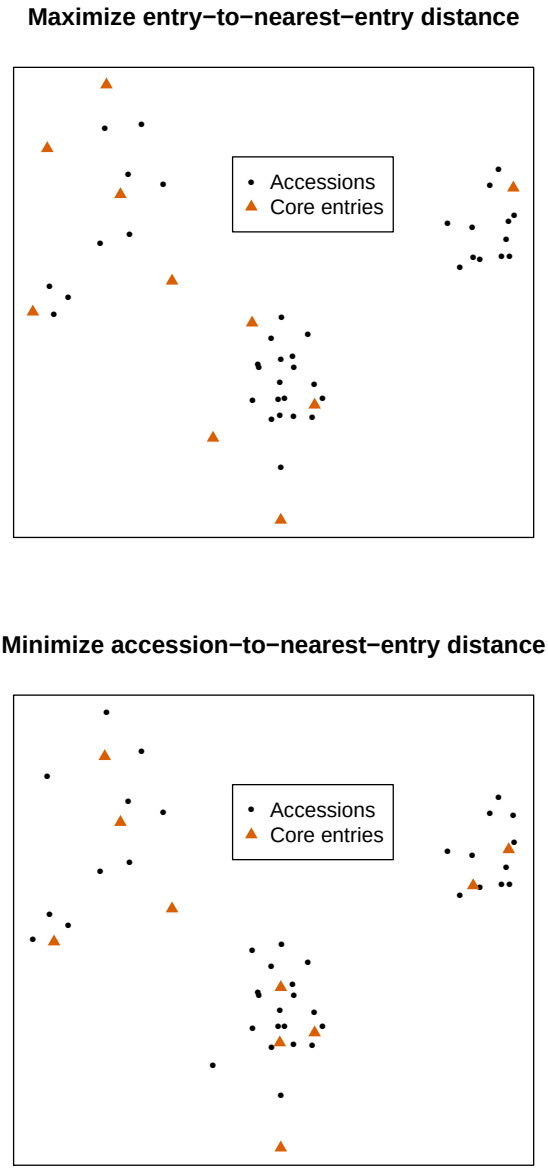


Figure 4.1: Multi-dimensional scaling plots of cores obtained when maximizing the entry-to-nearest-entry distance (top) as compared to minimizing the accession-to-nearest-entry distance (bottom) for a small generated dataset with 50 accessions, from which 10 are selected. The former objective tends to include accessions at cluster edges to maximize within-core diversity, while the latter favours accessions near cluster centers to optimally represent all individual data points.

where F_i is the i-th included measure and $0 < \alpha_i < 1$ is the weight assigned to this objective, with $\sum_{i=1}^{k} \alpha_i = 1$. In case of a measure F_i that is to be minimized, such as A-NE, it is transformed into a maximization objective $F'_i = -F_i$ when it is included in the weighted index. By default, the individual measures are also automatically normalized to $[0, 1]$ following the procedure described in section 2.3.1, to ensure a fair balance between the included objectives, independent of their original range.

4.3.3 *Core sampling algorithms*

Core Hunter 3 uses the random descent and parallel tempering local search algorithms, described in detail in section 2.2.4 and implemented in the JAMES framework (see chapter 3), to optimize the chosen evaluation measure or weighted index for a fixed core size. Both of these algorithms were also included in CH2, and based on the findings in this chapter CH3 defaults to the parallel tempering algorithm. The chosen algorithm is executed until a certain stop condition has been satisfied. Core Hunter 3 allows to specify an absolute runtime as well as a maximum time without finding any further improvement over the current selection.

When combining multiple objective functions in a weighted index, and given that normalization is enabled as in the default setting, the best solution in terms of each individual measure is approximated prior to the main optimization. For this purpose, we apply a simple random descent search to optimize each individual objective—a potentially rough approximation is usually sufficient to determine appropriate normalization ranges. These preliminary searches are executed in parallel, to reduce the computational overhead due to normalization, and with the same stop conditions as those specified for the main search.

Variable-size core sampling is no longer supported because the provided evaluation measures are not generally applicable to compare cores of different sizes. For example, reducing the core size artificially increases dissimilarity between selected accessions while adding more accessions always yields a more representative core. Also, while CH1 and CH2 preferred the smallest of two cores with the same value, minimizing the core size may not always be desired depending on the purpose of the core. We therefore believe that fixed- and variable-size core sampling should be treated as separate problems.

Random descent

This basic algorithm starts with a random selection of the desired size and then iteratively tries to improve its quality by swapping a randomly chosen selected and unselected accession. The obtained neighbouring solution is accepted as the new current solution if it has a higher quality, otherwise another move is tried starting from the same solution. For a detailed description, see section 2.2.4.

Parallel tempering

This more advanced algorithm, also known as replica exchange Monte Carlo (REMC), consists of multiple cooperating local searches that are executed in parallel. Each search performs the same procedure as random descent but may also accept inferior moves to be able to escape from local optima. The searches are assigned equally-spaced temperatures in a given range, where a higher temperature leads to a higher probability to accept inferior moves. Searches with similar temperature periodically exchange their current solutions to push the most promising selections towards the coolest searches for convergence, and the worst solutions towards the hottest searches to escape from local optima. For formulas and a detailed description of the algorithm, see section 2.2.4. The parallel tempering algorithm used by Core Hunter 3 consists of 10 search replicas with a temperature range of $[10^{-8}, 10^{-4}]$, and uses the same single-swap neighbourhood as the random descent procedure described above.

4.3.4 *Comparison with GDOpt and SimEli*

Because the original GDOpt implementation described by Odong et al. (2011) seemed needlessly complex we used a different algorithm based on the same idea, as implemented in the R function `pam`. This function largely follows the algorithm of Kaufman and Rousseeuw (1990) to partition data around automatically identified medoids. The number of clusters was chosen equal to the desired core size and the returned medoids were selected as core accessions. Aiming for the most fair comparison, we also reimplemented SimEli in R considering both elimination criteria suggested by Krishnan et al. (2014). In each step, one of the two most similar accessions was eliminated by maximizing either the average distance to the remaining accessions (SimEli-A-RA) or the expected heterozygosity of the reduced collection (SimEli-HE), until the desired core size was obtained.

	Random descent	Parallel tempering
Rice	0.150 ±2.51e-4	0.151 ±4.69e-6
Coconut	0.575 ±5.63e-4	0.576 ±3.18e-5
Maize	0.433 ±4.26e-4	0.435 ±1.55e-4
Pea	0.333 ±1.59e-3	0.339 ±5.61e-4

Table 4.1: Comparison of random descent and parallel tempering algorithms when maximizing the entry-to-nearest-entry criterion (E-NE). Mean values and standard deviations are reported for 10 independently sampled core collections, with a runtime limit of five minutes.

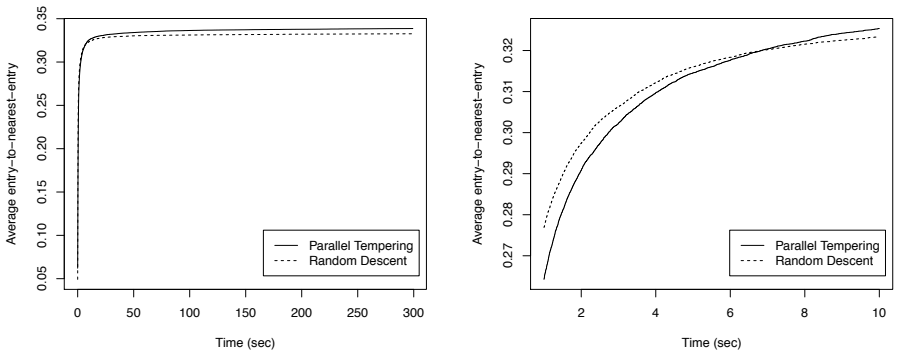


Figure 4.2: Convergence curves for the large pea dataset showing the E-NE value of the best found solution at each point in time during execution of the random descent and parallel tempering algorithms, averaged over 10 independent runs. The left plot reports the progress during the entire run, with a runtime of five minutes, while the right plot is zoomed in on the point where both curves intersect.

4.4 RESULTS

4.4.1 *Optimizing E-NE and A-NE*

We sampled 10 cores from each dataset using the random descent and parallel tempering algorithms configured to maximize E-NE with a runtime limit of five minutes. Table 4.1 shows mean values and standard deviations of the obtained cores. The results indicate that parallel tempering is able to construct cores with slightly higher E-NE values than random descent and reduces the variability across independently sampled cores, often by one or more orders of magnitude, although variability is already quite low when using random descent. Figure 4.2 displays the convergence curves of both algorithms, again averaged over 10 runs, for the pea dataset which

is the largest of the four considered datasets. These plots show that both algorithms are able to iteratively improve an arbitrarily bad random selection to reach a high E-NE value. Again, we see that parallel tempering yields a slightly higher E-NE value (left), but also that it is somewhat slower than random descent (right). Still, after about seven seconds, parallel tempering catches up with random descent, after which it keeps improving the quality of the core. We performed these experiments only for the E-NE measure but assume that our findings also hold for A-NE due to the very similar composition of both criteria. Therefore, all following CH3 results were obtained with the parallel tempering algorithm.

4.4.2 *Comparison with Core Hunter 2*

To assess whether maximizing E-NE indirectly also yields a high minimum distance (DMIN) between selected accessions we compared the results of CH3 and CH2. We configured CH2 to maximize a weighted index including both average and minimum pairwise distance, with equal weight, and CH3 to maximize E-NE. Both algorithms were terminated when no improvement was found during the last 10 seconds. Table 4.2 reports average E-NE, DMIN and execution time for 10 independent samples, obtained with both methods, for each dataset except the rice collection because CH2 cannot sample cores based on phenotypic traits.

For all three datasets, CH3 yields higher E-NE and DMIN than CH2. However, a detailed inspection of the output of CH2 (not shown) revealed that the LR replica—included in the MixRep search used by CH2—did not always complete before CH2 was terminated. As mentioned before, the LR search is a constructive heuristic that starts with an empty selection and iteratively adds accessions until the desired core size has been reached, and was specifically included in CH2 to construct cores with high minimum distance (De Beukelaer et al., 2012). Therefore, we repeated the CH2 experiments with an absolute runtime limit that was empirically determined per dataset to ensure that the LR replica terminated in each run (CH2L). Especially for the large pea dataset significantly more time was needed in this configuration, due to the quadratic time complexity of the LR replica. Table 4.2 shows that CH2L is indeed able to construct cores with much higher minimum distance than CH2 and also slightly outperforms CH3 in terms of this measure, although differences in minimum distance obtained with CH2L and CH3 are at most 4%. Moreover, CH3 still yields the highest-quality core collections in terms of the more elaborate E-NE criterion, and is faster for large datasets.

	E-NE	DMIN	Time (s)
Coconut			
CH2	0.552 ±3.53e-2	0.501 ±9.76e-2	27.6 ±06.0
CH3	**0.576** ±9.35e-5	0.540 ±0.00e-0	37.5 ±07.9
CH2L	0.569 ±5.91e-4	**0.548** ±0.00e-0	31.0 ±00.1
Maize			
CH2	0.416 ±1.52e-2	0.396 ±2.46e-2	78.3 ±10.6
CH3	**0.435** ±2.70e-4	0.409 ±3.05e-3	74.3 ±26.5
CH2L	0.429 ±5.00e-4	**0.415** ±1.11e-3	78.6 ±02.0
Pea			
CH2	0.219 ±1.49e-3	0.000 ±0.00e-0	85.6 ±04.5
CH3	**0.338** ±1.04e-3	0.287 ±1.34e-2	154.1 ±49.7
CH2L	0.325 ±8.21e-4	**0.297** ±0.00e-0	802.3 ±00.8

Table 4.2: Comparison of Core Hunter 2 and 3. CH2 maximizes a weighted index including average and minimum pairwise distance, with equal weight, while CH3 maximizes E-NE. Mean E-NE, DMIN, runtime, and corresponding standard deviations are reported for 10 independent executions. The highest obtained E-NE and DMIN value per dataset is shown in bold. CH3 was terminated when no improvements were found during 10 seconds. For CH2, two alternatives were considered: (a) the same stop condition as for CH3 (CH2); and (b) an absolute runtime limit that was empirically determined per dataset to ensure that the LR replica of MixRep terminated in each run (CH2L).

4.4.3 *Comparison with GDOpt and SimEli*

We approximated the Pareto front (see section 2.3) obtained by Core Hunter 3 when simultaneously optimizing E-NE, and either A-NE or HE with varying weights $\alpha_1 \in [0, 1]$ and $\alpha_2 = 1 - \alpha_1$, respectively, and compared the results with those obtained by GDOpt and SimEli. Note that A-NE is to be minimized, while E-NE and HE are maximized. As before, CH3 was terminated when no improvement was found during 10 seconds. Figure 4.3 shows that GDOpt and CH3 are able to construct representative cores with low A-NE, which is not the case for SimEli. In fact, all cores sampled by SimEli have a worse A-NE value than those obtained by GDOpt and CH3, even when the latter is configured to maximize E-NE only. On the other hand, SimEli scores much better than GDOpt in terms of di-

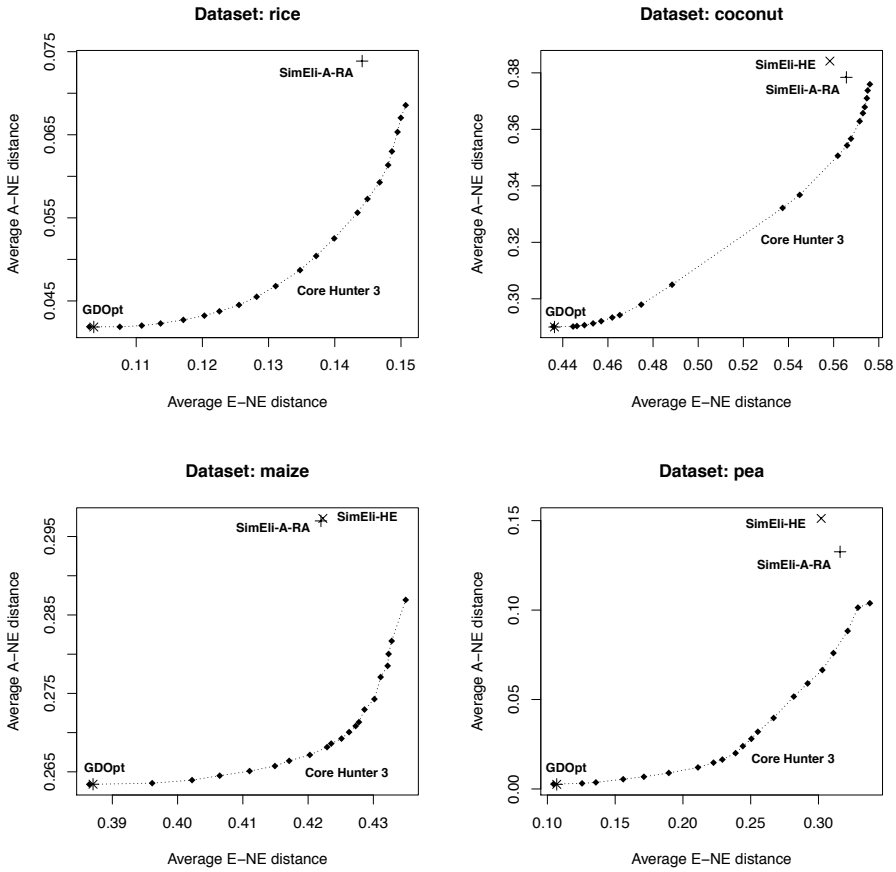


Figure 4.3: Simultaneous optimization of entry-to-nearest-entry (E-NE) and accession-to-nearest-entry (A-NE) distance. These Pareto front approximations for Core Hunter 3 were obtained by sampling cores with varying weights $\alpha_1 \in [0, 1]$ and $\alpha_2 = 1 - \alpha_1$ assigned to the E-NE and A-NE measures, respectively, with a step size of 0.05. The quality of the cores constructed by CH3 is compared with those found by GDOpt and SimEli, in terms of both criteria. All reported values are averages of 10 independently sampled cores with the same settings.

versity (high E-NE). Still, Core Hunter 3 is able to find cores which simultaneously have a higher diversity and are more representative than those obtained with SimEli. For the maize dataset, SimEli-A-RA and SimEli-HE found cores of similar quality, while for the coconut and pea dataset SimEli-A-RA showed to be preferred in terms of both E-NE and A-NE. For the rice dataset, SimEli-HE was not included in the comparison because expected heterozygosity can be evaluated for marker data only. Figure 4.4 shows that GDOpt yields cores with significantly lower HE than any of the other methods. SimEli performs better in this respect, especially SimEli-HE, but as before Core Hunter 3 is able to simultaneously improve over SimEli in terms of both objectives (E-NE and HE value).

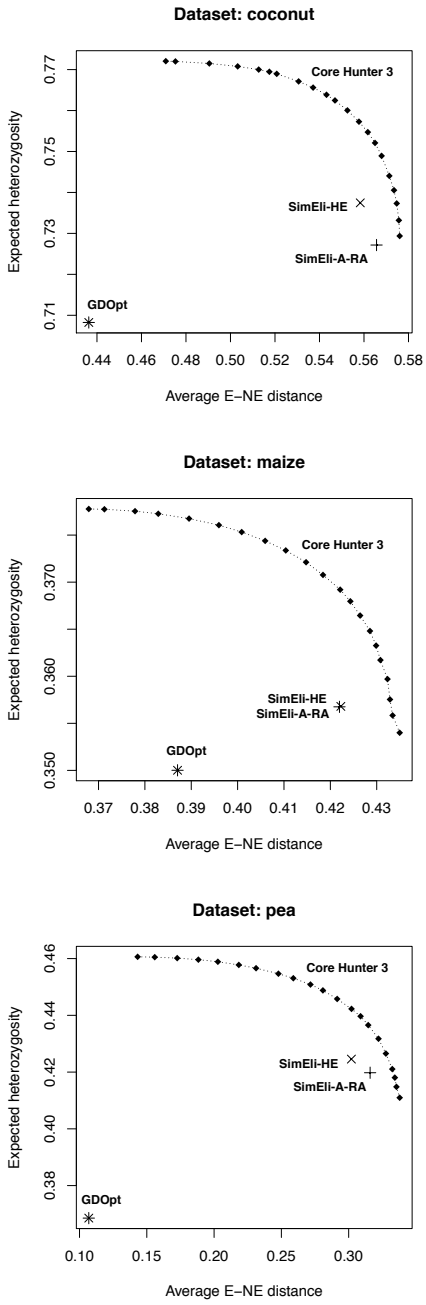


Figure 4.4: Simultaneous maximization of entry-to-nearest-entry distance (E-NE) and expected heterozygosity (HE). These Pareto front approximations for Core Hunter 3 were obtained by sampling cores with varying weights $\alpha_1 \in [0, 1]$ and $\alpha_2 = 1 - \alpha_1$ assigned to the E-NE and HE measures, respectively, with a step size of 0.05. The quality of the cores constructed by CH3 is compared with those obtained by GDOpt and SimEli, in terms of both criteria. All reported values are averages of 10 independently sampled cores with the same settings. The rice dataset is excluded here because expected heterozygosity can be evaluated for genotypic data only.

	Rice	Coconut	Maize	Pea
GDOpt	14.9	7.1	91.2	350.1
SimEli-A-RA	7.6	7.5	11.5	514.7
SimEli-HE	-	15.9	78.0	502.3
CH3 E-NE	45.8	37.5	74.3	154.1
CH3 A-NE	74.6	55.7	133.1	86.7
CH3 HE	-	16.6	40.2	62.8

Table 4.3: Average execution times (seconds) of GDOpt, both SimEli implementations, and CH3, for 10 independent samples from each dataset. Three configurations are considered for CH3: (a) maximize E-NE; (b) minimize A-NE; and (c) maximize HE.

Average execution times of GDOpt, SimEli and CH3 (configured to optimize E-NE, A-NE or HE) are reported in table 4.3. Core Hunter 3 was slower than GDOpt and SimEli for the rice and coconut datasets. For the maize dataset CH3 was faster than GDOpt and SimEli-HE when maximizing HE or E-NE but slower when minimizing A-NE and always slower than SimEli-A-RA. Finally, for the pea dataset, CH3 was notably faster than both GDOpt and SimEli. Core Hunter 3 was also consistently faster when maximizing HE as compared to the configurations in which E-NE or A-NE were optimized.

4.5 DISCUSSION

Depending on the purpose of a core collection, there are many possible ways to evaluate its quality. Distance-based measures are attractive because they are intuitive to understand and can capture both diversity within the core as well as representativeness of the individual accessions from the full collection. However, pairwise distances need to be aggregated in an appropriate way to evaluate the selected core. Although many studies and methods have used average pairwise distance to assess the diversity in the core, it is known that a high average does not guarantee that all selected accessions are sufficiently different from each other (De Beukelaer et al., 2012; Odong et al., 2013). Maximizing this criterion tends to overrepresent the extremes of the distribution.

Core Hunter 2 addressed this issue by maximizing the minimum in addition to the average pairwise distance, using a complex mixed replica method (MixRep) consisting of different cooperating algorithms (De Beukelaer et al., 2012). Previously, the original Core Hunter software successfully applied the parallel tempering al-

gorithm to maximize average distance and allelic richness, but such local search is not well suited to optimize minimum distance because this specific measure is very sensitive to the precise selection. Similar cores may have highly different values, while at the same time very different cores may have a similar or even the same minimum distance. This makes it difficult for a local search to find a way from a randomly generated selection to a high-quality core. In particular, for a given current solution, many possible modifications may not affect the minimum distance, meaning that the search has no clue as to whether these modifications may eventually lead to an improved solution. To smooth out the objective function, Core Hunter 2 could be configured to maximize a combination of average and minimum distance. Also, a constructive heuristic (LR) was included in the MixRep algorithm, which iteratively extends an initially empty selection and can therefore more easily maintain a high minimum distance between currently selected accessions. On the downside however, such constructive approach may become slow when sampling from large collections, as compared to a local search that already starts with a random selection which is subsequently further improved.

Odong et al. (2013) later suggested to instead maximize the average entry-to-nearest-entry (E-NE) distance. This criterion takes all accessions into account and can therefore presumably be more effectively optimized with local searches as compared to minimum distance, but still focuses on maintaining a high distance between each pair of closest accessions, which avoids overrepresentation of extreme values, in contrast to average pairwise distance. Therefore, in Core Hunter 3, the minimum distance measure was replaced by the newly proposed E-NE criterion. Another new measure was also included to sample cores that maximally represent all individual accessions from the full collection by minimizing the average accession-to-nearest-entry (A-NE) distance, again as proposed by Odong et al. (2013).

We assessed whether the new E-NE measure can indeed be effectively optimized with fast local searches, in an attempt to avoid the complexity and potential slowness of the MixRep algorithm. We showed that even a very basic stochastic hill-climber (random descent) can already construct cores with high E-NE value and little variability in quality across independent samples. Still, the value of the core is further improved, and variability further reduced, when using the more advanced parallel tempering algorithm. Since parallel tempering takes advantage of modern multi-core CPU architectures, by executing replicas in parallel, the associated computational overhead is limited. In our experiments, for the large pea dataset with over 4000 accessions, parallel tempering already

reached a higher E-NE value than random descent after only 7 seconds. We thus conclude that parallel tempering is preferred, and that more complex algorithms are not needed to optimize E-NE, and assume that the same conclusion holds for A-NE due to the very similar composition of both measures. Therefore, Core Hunter 3 uses parallel tempering by default, which is also known to effectively optimize the other measures that were already included in Core Hunter 2 (De Beukelaer et al., 2012). In addition, a fast mode is provided in which the basic random descent algorithm is applied, in case execution time is critical, but it was not used here.

The E-NE criterion was introduced because maximizing average distance may lead to cores in which too similar accessions are included, and maximizing minimum distance is too complex to be solved with simple and fast local searches. To validate the effectiveness of this alternative evaluation measure, we assessed whether maximizing E-NE indirectly also yields a high minimum distance. A comparison with Core Hunter 2, configured to sample cores with high average and minimum distance, revealed that maximizing E-NE indeed also leads to a high minimum distance. The minimum distance obtained with CH3 is slightly lower than for CH2, but more importantly CH3 yields higher E-NE values because it actively optimizes this criterion. As the minimum distance captures less information about the core than the E-NE criterion, we believe that it is better to focus on maximizing E-NE when aiming to construct a core with maximum diversity.

For large datasets, CH3 showed to be notably faster than CH2, due to the quadratic time complexity of the LR replica (De Beukelaer et al., 2012). In contrast to all other searches included in MixRep, LR does not start from a random selection that is iteratively improved, but from an empty selection to which accessions are added until the desired core size is reached. This means that it only produces useful results if given enough time to complete. Therefore, a potential issue of CH2 is that the user is responsible to set an appropriate time limit that allows the LR replica to complete, when aiming at a high minimum distance. Also, it may be confusing that there is a possibly large time gap between the last improvement found by the other replicas and that obtained when the LR replica has finished. In this respect, CH3 is more user-friendly because it uses a well-known local search algorithm that gradually improves the average E-NE value of the selected core. Large gaps between significant improvements are not expected which makes it much easier to determine an appropriate time limit and even more so to use a more convenient stop condition such as a maximum time without finding an improvement, in which case the execution time automatically adapts to the size of the dataset.

One of the main advantages of Core Hunter 3 and previous versions is its flexibility. While other methods are often developed for a specific purpose such as maximizing diversity, representativeness, or allelic richness, Core Hunter is suited for each of these as it includes a variety of evaluation measures that can directly be optimized and, if desired, combined in a weighted index. We compared CH3 with GDOpt, designed to maximize representativeness, and SimEli, where the elimination criterion was chosen either to maximize diversity (SimEli-A-RA) or expected heterozygosity (SimEli-HE). Core Hunter was configured to optimize a weighted index including E-NE and either (a) A-NE (figure 4.3); or (b) HE (figure 4.4), with varying weights, in order to approximate the corresponding Pareto front.

The results showed that, as expected, GDOpt is especially suited to construct cores that optimally represent all accessions from the entire collection (low A-NE), as it was specifically developed for this purpose. It is interesting to note that the effectiveness of our simple version of the GDOpt strategy suggests that the original implementation by Odong et al. (2011) is indeed needlessly complex. On the other hand, SimEli scores much better than GDOpt in terms of diversity (E-NE) and allelic richness (HE). From the two considered elimination criteria, SimEli-HE resulted in the highest allelic richness, while SimEli-A-RA showed to be most suited to maximize diversity (E-NE). Again, this was expected and confirms that the SimEli method can be adjusted to some extent by using an appropriate elimination criterion depending on the purpose of the core collection. However, Core Hunter 3 found cores that are simultaneously more diverse (E-NE), and either are more representative (A-NE) or have a higher allelic richness (HE), than those obtained by SimEli. In addition, CH3 was able to construct equally representative cores as GDOpt, and thus combines and improves over the advantages of both other methods.

A comparison of execution times showed that CH3 needs less time to optimize HE as compared to E-NE and A-NE. This is not surprising, as it is known that allelic richness can also be effectively maximized with a basic stochastic hill-climber (De Beukelaer et al., 2012). Since we showed that the more advanced parallel tempering algorithm is preferred to optimize E-NE and A-NE, it is clearly more difficult to find cores with high E-NE or low A-NE than to maximize allelic richness. Also, CH3 was somewhat slower than GDOpt and SimEli for smaller datasets, but faster for the large pea dataset. Here, the main advantage of Core Hunter is again its flexibility. For example, the runtime of SimEli is purely determined by the dataset and core size. When sampling a small core from a large collection, many accessions need to be eliminated, and find-

ing the two most similar accessions in each step as well as deciding which one to eliminate is a time consuming process. In contrast, the runtime of Core Hunter can be adjusted by using an appropriate stop condition, which provides a convenient quality-runtime tradeoff. It is possible to limit the total execution time, but in our experiments we used an adaptive condition that terminated the search when no more improvement was found during 10 seconds. This automatically adjusts the runtime to the size of the dataset, to some extent, while still offering sufficient flexibility, and largely influenced the execution times of Core Hunter 3 in our experiments. Of course, we may be able to further increase the quality of the core collections sampled from any of our datasets by allowing a longer runtime, but with the current settings Core Hunter already outperforms the other sampling algorithms, whose execution time cannot be controlled.

Overall, each of the tested methods showed to be very fast for datasets including at least several thousands of accessions. We are therefore convinced that execution time will not be an issue in practical applications, regardless of the chosen algorithm. Still, Core Hunter is the only one whose runtime can be controlled in various ways. The resulting quality-runtime tradeoff allows users to limit the execution time for large datasets if needed, for example when quickly exploring various sampling settings. In addition, and perhaps even more interestingly, it also allows to more thoroughly explore the solution space if more time is available, which may yield better cores. Note that although we did not experiment with genotypic datasets with several tens or hundreds of thousands of markers, these can easily be dealt with by, for example, precomputing a distance matrix so that only the number of accessions affects the performance of Core Hunter.

4.6 CONCLUSIONS

In this chapter we introduced Core Hunter 3 (CH3) and showed that it constructs core collections with high diversity (high average entry-to-nearest-entry distance; E-NE) and which maximally represent the individual accessions from the entire collection (low average accession-to-nearest-entry distance; A-NE) using flexible and fast local search algorithms. By default, the parallel tempering algorithm is used to optimize these and other evaluation measures.

Version 3 improves over Core Hunter 2 (CH2) in multiple ways. Core Hunter 3 is able to find cores with a higher E-NE value, within less time for large datasets, which also have a high minimum dis-

tance, and without the need for a more complex algorithm like the mixed replica search from CH2. In addition, CH3 finds similar and often better cores as compared to GDOpt and SimEli, which were reported to outperform CH2 in terms of E-NE and A-NE. In particular, CH3 can create equally representative cores as GDOpt, which was specifically designed for this purpose, while at the same time being able to construct cores that are simultaneously more diverse, and either are more representative or have a higher allelic richness, than cores obtained with SimEli.

As in previous versions, one of the main strengths of Core Hunter is its high flexibility. The applied local search algorithms are not confined to a specific evaluation measure and new criteria can easily be introduced and optimized without the need to alter the underlying algorithms. Moreover, multiple criteria can be simultaneously optimized and the execution time is controlled by the user through various stop conditions. We therefore believe that, from all available methods, Core Hunter is the most broadly applicable core subset selection tool with a lot of opportunities to be further extended.

ACKNOWLEDGEMENTS

We would like to thank Nathan Sinnesael who performed preliminary experiments supporting the development of Core Hunter 3.

IMPLEMENTATION AND HARDWARE

Core Hunter 3 has been reimplemented from scratch in Java 8 using the JAMES framework (v1.2) presented in chapter 3, and was executed from R through the package `corehunter`. GDOpt, SimEli and all experiments were implemented in R v3.3.1 (R Core Team, 2015) and executed on a computing server with two 10-core Intel E5-2660v3 (2.6 GHz) processors and 128 GB RAM.

5

LONG-TERM GENOMIC SELECTION STRATEGIES

SUMMARY

Long-term genomic selection (GS) requires strategies that balance genetic gain with population diversity, to sustain progress for traits under selection and to keep diversity for future breeding. In a simulation model for a recurrent selection scheme we provide the first head-to-head comparison of two such existing strategies: genomic optimal contributions selection (GOCS) that limits realized genomic relationship among selection candidates, and weighted genomic selection (WGS) that upscales rare allele effects.

Compared to GS both methods provide the same higher long-term genetic gain and a similar lower inbreeding rate despite that some inherent limitations were observed. GOCS does not control a specific component of the inbreeding rate that is linked to trait selection, and therefore does not strike the optimal balance between genetic gain and inbreeding. As such this method becomes less effective throughout the breeding scheme and particularly so at the beginning where genetic gain and diversity may not yet be competing. For WGS the truncation selection approach proved suboptimal to manage rare allele frequencies among the selection candidates.

To overcome these limitations we introduce two new set selection methods that maximize a weighted index balancing genetic gain with controlling expected heterozygosity (IND-HE) or maintaining rare alleles (IND-RA), and show that these outperform GOCS and WGS in a nearly identical way. While requiring further testing, we believe that the inherent benefits of the IND-HE and IND-RA methods will transfer from our simulation framework to many practical breeding settings, and are therefore a major step forward towards efficient long-term genomic selection.

5.1 INTRODUCTION

Genomic selection (GS) was initially proposed by Meuwissen et al. (2001) and uses existing phenotypes and marker information to obtain breeding values for untested selection candidates (see sec-

tion 1.2.3). With cheap high density genotyping becoming available, GS was introduced in many animal and plant breeding programs over the last decade. The added value of GS is generally attributed to accelerated breeding by shortened generation intervals, and to higher selection accuracy especially for traits that are difficult to observe (Daetwyler et al., 2013; Hayes et al., 2009; VanRaden et al., 2009; Wiggans et al., 2011). Unfortunately, GS also accelerates loss of genetic diversity due to the quick fixation of large effect loci, and likely also due to the higher selection accuracy of individuals with close relationship to the training set (Badke et al., 2014; Wientjes et al., 2013). Because this loss of diversity limits long-term gain for the trait under selection (Jannink, 2010) and also jeopardizes future breeding for other traits, we need GS strategies that balance genetic gain with diversity.

Animal breeders have widely adopted optimal contributions selection (OCS; Meuwissen 1997) to manage population diversity during long-term selection. OCS maximizes genetic gain under a predefined pedigree-based inbreeding rate by calculating the optimal contribution of all selection candidates to the next generation by means of Lagrangian multipliers. Since its introduction, OCS has been considerably refined to accommodate operational breeding constraints such as restricting the number of individuals contributing to the next generation, and imposing upper or lower limits on how much an individual contributes. Meuwissen (2002) manages these additional constraints with an iterative heuristic wrapped around the original mathematical solution that removes individuals with a too low contribution and truncates contributions exceeding the maximum, while repeatedly re-optimizing the remaining contributions (Woolliams et al., 2015). Alternatively, the operational constraints can be modelled directly using semidefinite programming, which may provide slightly higher gains at the cost of a more complex problem formulation (Ahlinder et al., 2014; Pong-Wong and Woolliams, 2007).

Lagrangian multipliers can be used to optimize a continuous function subject to one or more equality constraints, given that both the objective function and all constraints have continuous first-order partial derivatives. For details about this method, we refer to Bertsekas (2014).

A different strategy is to leave the strict constrained optimization framework and maximize a weighted index that balances genetic gain and inbreeding (Carvalheiro et al., 2010; Clark et al., 2013). Optimizing this index with general purpose metaheuristics such as a differential evolution algorithm (Storn and Price, 1997) allows to easily accommodate additional constraints and objectives, trading optimality of solutions for flexibility. This allowed Kinghorn (2011) to move from assigning individual contributions to identifying optimal mating pairs.

The OCS strategy can also handle typical constraints in plant breeding applications that often have a fixed number (n) of selected

individuals contributing equally to the next cycle. Most straightforward is to use the heuristic of Meuwissen (2002) and set both the minimum and maximum contribution to $1/n$. This approach showed to work well, as a more direct and theoretically exact solution with a branch-and-bound algorithm (Mullin and Belotti, 2016) provided only marginally better results that do not justify the additional complexity and computation time. As such OCS has become a well-established method that can be used in many if not all practical animal and plant breeding applications.

In the genomics era the availability of marker data for the selection candidates allowed a further extension of OCS: genomic OCS or GOCS (Sonesson et al., 2012). In GOCS the realized genomic relationship matrix $\mathbf{G}$, computed from marker data, replaces the expected additive relationship matrix $\mathbf{A}$, inferred from the pedigree, in the OCS formulas (Woolliams et al., 2015). Intuitively this makes a lot of sense as due to selection pressure we expect the realized relationships to differ from the pedigree-based average as well as being unequally distributed across the genome. For these reasons GOCS is the current method of choice for controlling inbreeding in a GS context.

Avoiding the loss of favourable rare alleles has also received attention in view of increasing long-term selection gain. For GS, Jannink (2010) proposed a weighted strategy (WGS) in which the effect of rare favourable alleles is amplified following theory by Goddard (2009). Extensions of WGS were proposed with additional parameters to balance short- and long-term gain (Sun and VanRaden, 2014) or to dynamically reduce the focus on rare favourable alleles over a fixed time horizon (Liu et al., 2015). Many other weighting schemes could be explored, including one derived from the theory of Liu and Woolliams (2010) that determines the optimal QTL allele trajectory from its initial frequency to fixation, although in our own comparison based on simulated data these weights provided very similar results as standard WGS (H De Beukelaer, G De Meyer; personal communication).

Instead of amplifying allele effects in the selection index calculated for every individual it might be more effective to directly control rare allele frequencies in the set of selected individuals, following the approach of Li et al. (2008) to stack known QTL. These authors applied a differential evolution algorithm to maximize a weighted index including the summed QTL allele effects and one of several diversity measures, and found that avoiding loss of rare favourable alleles most effectively maximized long-term gain. As this approach has not yet been evaluated in a GS setting, the merit of maintaining rare favourable alleles for long-term genomic selection cannot be

fully appreciated. To our knowledge, strategies based on maintaining rare favourable alleles—or rare alleles in general—have not yet been directly compared to GOCS.

In this chapter, we provide a detailed comparison of several diversity management strategies in view of increasing long-term genomic selection gain and maintaining overall genetic diversity. We focus on a typical recurrent selection plant breeding scheme, with a fixed number of individuals selected in each cycle that equally contribute to the next generation. Through simulations, we first compare existing implementations of WGS and GOCS, and assess their relative improvement over standard GS. Next, we use a unified optimization framework that maximizes a weighted index containing breeding value and a well-chosen population diversity measure, to contrast WGS and GOCS to new alternative methods that address some of their inherent limitations. Based on the results, we discuss the pros and cons of the different selection strategies from both a practical and theoretical perspective.

5.2 MATERIALS AND METHODS

5.2.1 *Haplotypes, base populations and genetic trait architecture*

Through chromosome doubling, haploid sperm or egg cells can be transformed into homozygous "doubled haploids" and grown into a doubled haploid plant. This provides a single-generation shortcut towards homozygosity which otherwise requires about six generations of conventional inbreeding. Breeders like homozygous inbred lines because they are stable, uniform, and can easily be reproduced.

To serve as the backbone for the simulations we derived haplotypes from genotypes of 192 founder inbred lines from the Oregon State University winter barley breeding program (Blake et al., 2012) using the consensus map (Close et al., 2009) for marker positions, spanning a total of 1091 cM. The raw genotype dataset contained 2591 SNPs and was preprocessed to retain 2031 polymorphic SNPs at unique positions with more than 99% homozygous values (see appendix A.1). Haplotypes were inferred using Beagle (Browning and Browning, 2009; Browning and Browning, 2007) through synbreed (Wimmer et al., 2015). Following Sonesson et al. (2012) we positioned additional artificial identity-by-descent (IBD) markers with unique founders alleles at an equal distance of 10 cM on all chromosomes. These IBD markers were not used for selection or prediction, but only to evaluate inbreeding based on genomic identity-by-descent. We simulated 200 base populations by randomly mating the 192 founders, followed by doubled haploid (DH) creation. In each base population 1000 out of 2031 SNPs were randomly selected to be additive QTL for a complex trait and removed from the marker dataset. The remaining 1031 SNPs were used as genetic markers. QTL effects were sampled from a standard normal distribution. The residual trait variance was derived from the assumed

heritability and the observed genetic variance in the base population, and was kept fixed for the entire simulation run.

The heritability of a trait is the proportion of observed variance that is due to genetic variance. Breeders like highly heritable traits because then good genotypes consistently yield high-quality phenotypes. Unfortunately however, many traits have a quite low heritability, often somewhere in the range from 0.2 to 0.5, complicating the task of discriminating between the good, the bad, and the ugly in a breeding program due to noisy phenotypic observations.

5.2.2 *Genomic prediction*

We used Bayesian ridge regression to estimate marker effects based on the linear model

$$\mathbf{y} = \mu + \mathbf{X}\boldsymbol{\beta} + \mathbf{e}$$

where $\mathbf{y}$ is a vector of phenotypes, μ is the population mean, $\mathbf{X}$ is a design matrix containing the 0/1-coded DH marker data (individuals $\times$ markers), $\boldsymbol{\beta}$ is a vector of marker effects and $\mathbf{e}$ is a vector of random residuals. The model was fit using the R package BGLR (Campos and Pérez, 2015) with default prior distributions and initial values. The genomic estimated breeding value (GEBV) of prediction individuals with genotypes $\mathbf{X}$ was calculated as

$$\mathbf{GEBV} = \mathbf{X} \cdot \hat{\boldsymbol{\beta}}$$

where $\hat{\boldsymbol{\beta}}$ is the vector of estimated marker effects.

5.2.3 *Breeding program simulations*

For all four combinations of a low ($h^2 = 0.2$) and high ($h^2 = 0.5$) heritability as well as a small (TP = 200) and large (TP = 1000) initial training population we performed 200 simulations of the breeding program defined by Jannink (2010) for the duration of 30 cycles. Each simulation run starts from a different base population that serves as the initial selection population and as the training population (TP) to fit an initial genomic prediction (GP) model. In case of a large training population, the base population was complemented with 800 additional phenotyped individuals that were obtained from the founder dataset using the same procedure (as described above) but these were not considered candidates for selection. Input for the prediction model are the marker genotypes and phenotypes, inferred from the summed QTL effects and a random error sampled from a normal distribution with variance equal to the residual trait variance under the assumed heritability.

Using standard GS or alternative selection strategies (see below) 20 individuals are selected for random intermating followed by doubled haploid creation to generate 200 new selection candidates. In cycle 2 the same selection procedure is applied using the original GP model while in parallel the 200 selection candidates are phenotyped to augment the model for use in cycle 3 (Jannink, 2010). The process then iterates for 30 cycles (see figure 1.5).

For each simulation scenario, several variables were extracted and averaged over the 200 replicates. The first tracked variable is (cumulative) genetic gain, expressed as the increase in average true genetic value as compared to the base population. Prior to calculating gain, genetic values were normalized to $[-1, 1]$ based on the minimum and maximum attainable value over all possible genotypes. We also tracked the inbreeding rate (Falconer et al., 1996)

$$\Delta F = \frac{F_t - F_{t-1}}{1 - F_{t-1}}$$

where F_t is operationalized as the average expected homozygosity of either the artificial IBD markers (ΔF_{IBD}) or the actual SNP marker panel (ΔF_{IBS}):

$$F_t = \frac{1}{m} \sum_i \sum_j p_{ij}^2$$

where p_{ij} is the frequency of the j-th allele of the i-th marker—IBD or SNP—and m is the total number of the respective kind of markers. In this way, inbreeding is expressed as the relative decrease in expected heterozygosity based on either identity-by-descent (ΔF_{IBD}) or identity-by-state (ΔF_{IBS}). At selection population level we also tracked the number of favourable QTL alleles lost and their total effect, the mean QTL favourable allele frequency, and the number of SNP alleles lost in general.

5.2.4 *Standard and weighted genomic selection*

Both standard (GS) and weighted (WGS) genomic selection rank individuals according to their (w)GEBV and select the n candidates with the highest value. With WGS, marker effects are scaled to obtain weighted breeding values

$$\mathbf{wGEBV} = \mathbf{X} \cdot \begin{bmatrix} w_1 \hat{\beta}_1 \\ \vdots \\ w_n \hat{\beta}_n \end{bmatrix}$$

where $\hat{\beta}_i$ is the estimated effect of the i-th marker and w_i is the weight assigned to that marker. We used the weights

$$w_i = f_i^{-0.5}$$

as defined by Jannink (2010) where f_i is the favourable allele frequency of the i-th marker, in the selection population.

5.2.5 *Optimal contributions selection*

For GOCS (Sonesson et al., 2012; Woolliams et al., 2015) optimal contributions $\mathbf{c_t}$ are assigned to the selection candidates by maximizing the expected genetic level $\mathbf{g_{t+1}}$ in the next generation under a constraint that aims to realize a predefined target inbreeding rate $C_{t+1} = \Delta F_{target}$:

$$\begin{aligned} \max \quad & \mathbf{g_{t+1}} = \mathbf{c_t^T GEBV_t} \\ \text{with} \quad & C_{t+1} = \frac{\mathbf{c_t^T G_t c_t}}{2}. \end{aligned}$$

Here, $\mathbf{GEBV_t}$ is a vector with breeding values in generation t and the realized genomic relationship matrix $\mathbf{G_t}$ is used to constrain the expected inbreeding in the next generation. We followed VanRaden (2008) to calculate $\mathbf{G_t} = \mathbf{Z_t Z_t^T}/[2\sum p_j(1-p_j)]$ where $\mathbf{Z_t}$ contains reference allele counts relative to the population mean (Woolliams et al., 2015). As opposed to OCS where C_{t+1} is increased over generations to account for accumulated absolute inbreeding using the formula $C_{t+1} = 1-(1-\Delta F_{target})^t$ (Grundy et al., 1998) a fixed value $C_{t+1} = \Delta F_{target}$ is set over all generations in GOCS as, unlike the expected pedigree-based relationship matrix $\mathbf{A}$, the realized genomic relationship matrix $\mathbf{G}$ is naturally scaled relative to the population mean. More details are provided in appendix A.2.1.

To match the simulated breeding scheme that crosses n parents with equal contribution, we imposed a minimum and maximum contribution of $c_{min} = c_{max} = 1/n$, using the iterative heuristic of Meuwissen (2002). This algorithm discards individuals with a too low contribution and truncates those exceeding the maximum, while repeatedly re-optimizing the remaining contributions. Due to the operational constraint it is not always possible to achieve precisely the desired value for C_{t+1}. Therefore, at a certain iteration, our implementation of the applied heuristic may switch to minimizing realized genomic relationship to assign the remaining contributions, as explained in detail in appendix A.2.2.

5.2.6 *Unified set selection framework*

To obtain a truly fair comparison between existing strategies, and some new alternatives, we implemented them in a unified optimization framework that uses general purpose metaheuristics to identify optimal subsets. We selected a set of individuals by maximizing a weighted index

$$F(S) = (1-\alpha)\cdot V(S) + \alpha \cdot D(S)$$

where S is a subset of the selection candidates and $\alpha \in [0,1]$ is a weight used to balance genetic merit $V(S)$, defined as the average GEBV of the selection, and diversity $D(S)$. Both components are dynamically normalized to $[0,1]$ using the procedure described in section 2.3.1. We experimented with three different diversity measures, including minimization of the realized genomic relationship as used in OCS to control inbreeding (IND-OC):

$$D_{OC}(S) = -\frac{\mathbf{c_t}^T \mathbf{G_t} \mathbf{c_t}}{2}$$

where $\mathbf{c_t}$ is a vector with $1/n$ for each selected parent in S, and 0 for each remaining individual (Lindgren and Mullin, 1997). A second version (IND-HE) balances gain and expected heterozygosity, in an attempt to control the inbreeding rate when defined as the relative decrease in expected heterozygosity computed from the SNP markers (ΔF_{IBS}):

$$D_{HE}(S) = \frac{1}{m} \sum_{i=1}^{m} 2p_i(1-p_i)$$

where p_i is the minor allele frequency of the i-th marker, in the selected set S, and m is the number of markers. Finally, IND-RA directly manages rare alleles using a criterion similar to one that was previously shown to be effective in a context with known QTL and effects (Li et al., 2008):

$$D_{RA}(S) = \frac{1}{m} \sum_{i=1}^{m} \log(p_i)$$

where p_i and m are again the minor allele frequency and the number of markers, respectively. In contrast to Li et al. (2008) we consider all alleles and not only the favourable ones. The logarithm was truncated at

$$\log(0) := -(\log(n) + 1) \tag{5.1}$$

to avoid that all selections in which at least one allele becomes fixed would be incomparably evaluated at minus infinity (n being the selection size).

As optimization engine we used the parallel tempering local search metaheuristic as described in section 2.2.4, with ten replicas, a temperature range of $[10^{-8}, 10^{-3}]$, and a neighbourhood that randomly swaps one selected and unselected item in an attempt to improve the quality of the selection. The algorithm terminated when no improvement was found during five seconds. For normalization purposes, we composed the best possible selection in terms of breeding value only by selecting the n items with the highest individual

		IND		
	GOCS (C_{t+1})	**OC** (α)	**HE** (α)	**RA** (α)
Scenario 1	0.05	0.35	0.35	0.35
Scenario 2	0.02	0.65	0.35	0.35

Table 5.1: Considered parameter values when using the optimal contributions selection (OCS) method, and when maximizing a weighted index containing average breeding value and a specific diversity measure (IND-OC, IND-HE, IND-RA).

value. In addition, we used the same parallel tempering algorithm as for the main optimization procedure to approximate the best selection when maximizing diversity only—allowing no more than three seconds without finding any further improvement. These two extreme solutions were then used to determine the normalization ranges of $V(S)$ and $D(S)$, as described in section 2.3.1. Selection was implemented using the JAMES framework presented in chapter 3.

5.2.7 *Parameter values*

For GOCS, IND-OC, IND-HE and IND-RA we considered two values for the parameters C_{t+1} and α, respectively. First, we searched for the lowest value of C_{t+1} (highest α) that still yields at least the same short-term gain as WGS. Secondly, we determined parameter values that resulted in roughly the same observed inbreeding rate ΔF_{IBS} for these four methods. The optimal values for both scenarios (table 5.1) were determined empirically, through a grid search with $\alpha \in [0, 1]$ and a step size of 0.05 for IND-OC, IND-HE and IND-RA, and $C_{t+1} \in [0.01, 0.1]$ with a step size of 0.01 for GOCS.

5.3 RESULTS

5.3.1 *Simulation framework*

We compare several long-term selection strategies in the GS-based recurrent selection plant breeding scheme from Jannink (2010). Simulations were performed on a genome with 2031 SNPs allowing the positioning of a 1000 QTL quantitative trait while leaving the remaining 1031 SNPs (about 1 SNP per cM) to be used for selection and diversity management. Because for some selection strategies genetic gain was still observed beyond the 20 cycles considered by

Jannink (2010) we extended the scheme to 30 cycles to fully appreciate the long-term dynamics.

In our simulation framework trait heritability ($h^2 = 0.2$ vs. $h^2 = 0.5$) had a major effect on the GS genetic gain (figure 5.1) with, as expected, a higher genetic gain plateau for the high heritability. We also observe significant inbreeding under GS which is more pronounced for ΔF_{IBS} as compared to ΔF_{IBD}. Varying the size of the training population (TP = 200 vs. TP = 1000) to build the initial genomic prediction model had less effect on genetic gain over time in general, and on the relative performance of the selection strategies in particular. Therefore, we provide the results for TP = 200 in appendix A.

5.3.2 *Weighted GS and genomic OCS*

We observe that irrespective of the trait heritability WGS slightly reduces the short-term genetic gain as compared to GS—by at most one cycle—to achieve a significantly higher long-term gain (figures 5.1 and A.1; left panel). This goes hand in hand with a general control of the inbreeding rate, with minor differences between ΔF_{IBS} and ΔF_{IBD} but with a clear dependency on the training population size and heritability, as higher inbreeding rates were observed in case of a low heritability and/or small TP. At the first cycle the inbreeding rate is exceptionally lower and even negative for ΔF_{IBS}.

Moreover, GOCS—with a constraint set at 0.05 to mimic the WGS short-term gain—performed very similar to WGS in terms of long-term gain as well (figures 5.1 and A.1; right panel). GOCS also reduced the inbreeding as compared to GS, and more consistently than WGS. In particular, GOCS provides more stable inbreeding rates across cycles, that are independent of the training population size and heritability. Yet, as opposed to WGS, the GOCS inbreeding rate ΔF_{IBS} is higher than ΔF_{IBD}. Also, the response during the first cycles is different, with GOCS gradually building up to a stable value while WGS only had a low spike at the first cycle. Most notably, over generations, the GOCS inbreeding rate consistently deviates from the constraint value of $C_{t+1} = 0.05$ that should match the inbreeding rate ΔF, despite that this constraint was always closely reached in the optimization routine (results not shown).

To explore possible underlying mechanisms causing the latter observation we also ran our simulations without genomic selection—i. e. with a random selection of 20 individuals in each cycle—as well as with a selection size of 50 instead of 20 (figure A.2). In

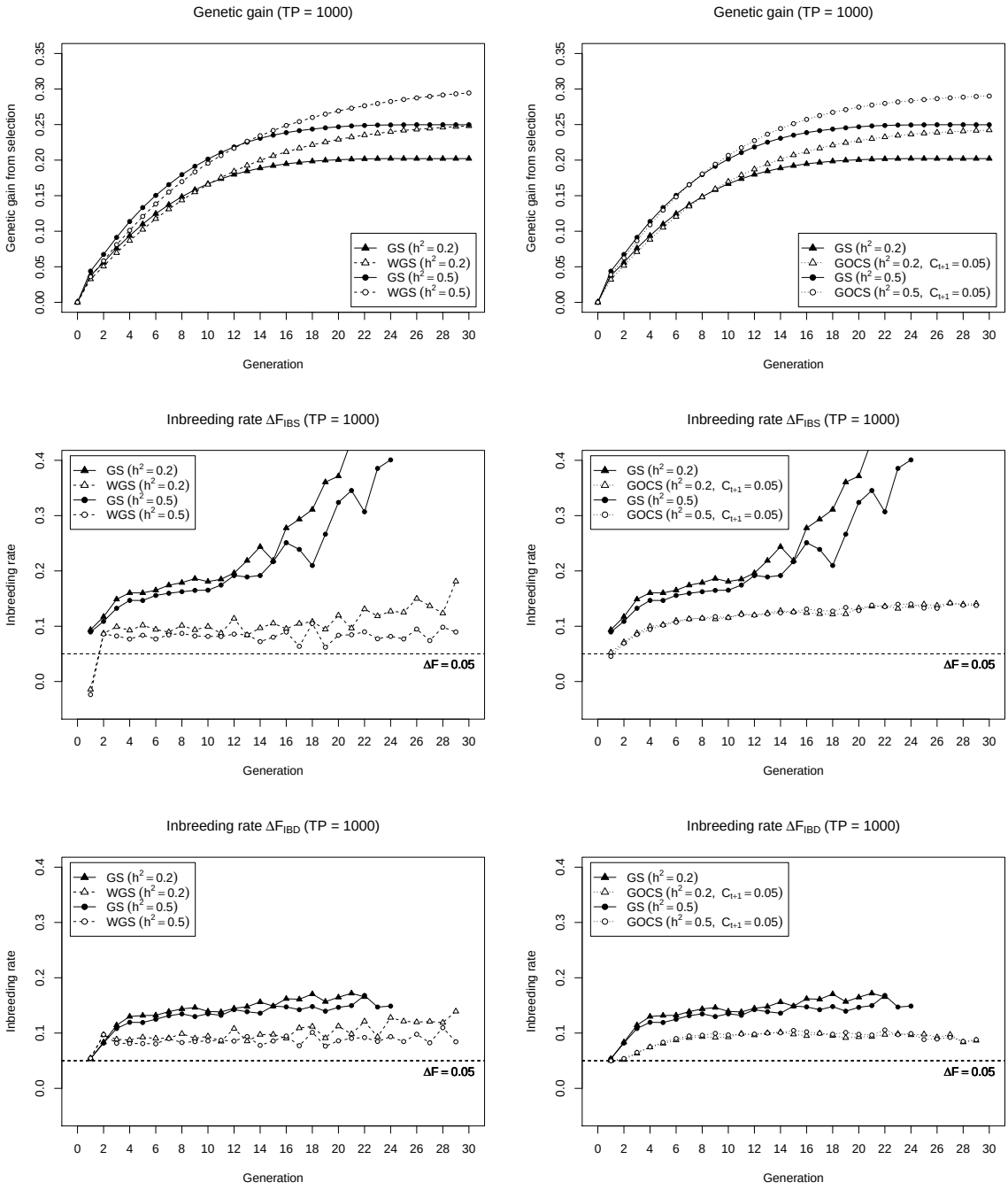


Figure 5.1: Cumulative genetic gain (top) and inbreeding rate (IBS: middle; IBD: bottom) for weighted genomic selection (WGS; left) and genomic optimal contributions selection (GOCS; right) as compared to standard genomic selection (GS). Results are reported for a low ($h^2 = 0.2$) and high ($h^2 = 0.5$) heritability with a large initial training population (TP = 1000) and are averages of 200 simulation runs. The inbreeding rates are reported until at least half of the simulation runs have lost all variability for the SNP marker panel used.

the absence of selection both ΔF_{IBS} and ΔF_{IBD} remained constant at 0.05, which is the expected drift (1/20) when randomly mating 20 doubled haploid plants. Exactly the same average value of 0.05 was observed when evaluating the GOCS criterion $\mathbf{c}_t^T \mathbf{G}_t \mathbf{c}_t/2$ in this setting. When selecting 50 individuals, GOCS moves towards the $\Delta F = 0.05$ plateau for ΔF_{IBD}, but not for ΔF_{IBS} which still exceeds the target inbreeding rate.

In summary we conclude that WGS and GOCS achieve similar long-term genetic gain but have different behaviour at the level of the inbreeding rate where in particular GOCS does not control inbreeding at the target level.

5.3.3 *Unified set selection framework*

In order to more directly compare selection strategies that control inbreeding or manage rare alleles we evaluated these in a unified optimization framework that maximizes a weighted index of average breeding value and a diversity measure chosen to either minimize realized genomic relationship (IND-OC), maximize expected heterozygosity (IND-HE), or retain rare alleles (IND-RA).

In a first scenario with GOCS and the index-based methods parameterized for a short-term genetic gain comparable to WGS (figure 5.2; left panel) IND-OC provides a genetic gain profile similar to GOCS and WGS, while IND-RA and IND-HE give clearly higher long-term gains. IND-OC also roughly parallels GOCS in terms of inbreeding rate (except for ΔF_{IBS} in the last few cycles) again with clearly higher values for ΔF_{IBS} as compared to ΔF_{IBD}. IND-RA and IND-HE give similar inbreeding rates below those of IND-OC and GOCS. In addition there is almost no difference between ΔF_{IBS} and ΔF_{IBD}, and in contrast to IND-OC and GOCS a strongly negative inbreeding rate ΔF_{IBS} is observed in the first generation. This pattern closely resembles that observed for WGS (figure 5.1) but with much less variability.

In a second scenario with GOCS and IND-OC parameterized so that the realized inbreeding rate ΔF_{IBS} does not exceed that of IND-RA and IND-HE during the entire simulation (figure 5.2; right panel) the higher long-term gain obtained by both GOCS and IND-OC comes with a major penalty on short-term gain, and is still outperformed by IND-RA and IND-HE. In this setting, with a less pronounced selection for the simulated trait, inbreeding rates ΔF_{IBS} and ΔF_{IBD} of GOCS and IND-OC are more similar and, although also more stable over time, still clearly deviate from the expected

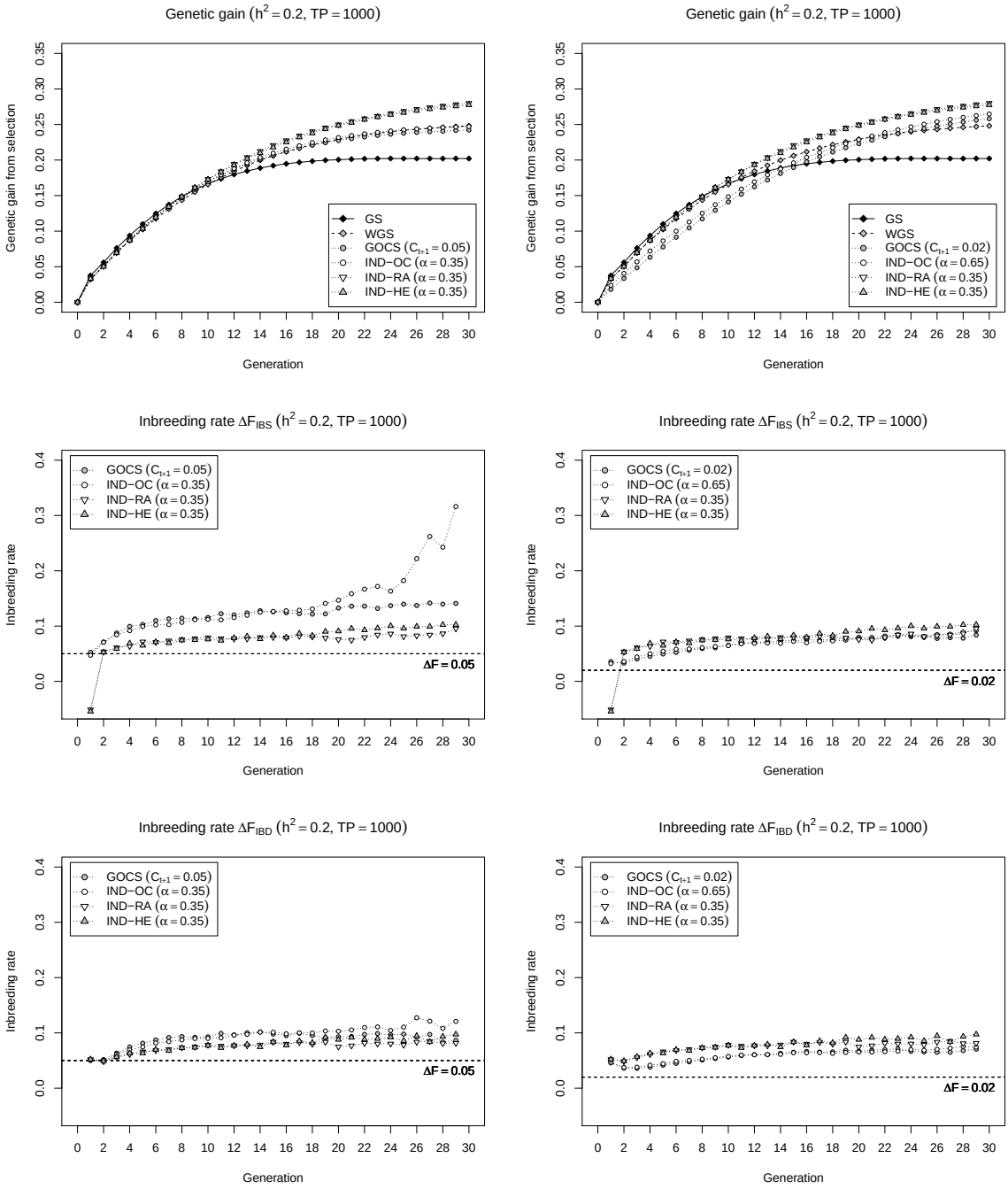


Figure 5.2: Cumulative genetic gain (top) and inbreeding rate (IBS: middle; IBD: bottom) of selection strategies that maximize a weighted index containing breeding value and a diversity measure chosen to control inbreeding (IND-OC, IND-HE) or to avoid loss of rare alleles (IND-RA). Results for GS, WGS, and GOCS are provided as a reference. For clarity, inbreeding rates of GS and WGS are omitted. Two scenarios were considered to set the parameters C_{t+1} and α: maintain the same short-term gain as WGS (left), or achieve a similar inbreeding rate ΔF_{IBS} (right). Results are reported for a low heritability ($h^2 = 0.2$) with a large initial training population (TP = 1000) and are averages of 200 simulation runs.

value of 0.02. Very similar results are observed for the small TP (figure A.3) and $h^2 = 0.5$ settings (figures A.4 and A.5).

Overall we conclude that IND-RA and IND-HE are roughly equivalent long-term selection strategies that outperform WGS and GOCS in our simulation framework.

5.3.4 *Drift and selection at locus level*

To better understand the underlying mechanisms operating in figure 5.2 we quantified the loss of favourable QTL alleles, the corresponding QTL effect, the increase of favourable QTL allele frequencies, and the number of SNP alleles lost in general (figure 5.3). In the strong short-term gain scenario, on the left panel, IND-RA and IND-HE retained clearly more favourable QTL alleles than WGS, GOCS, and IND-OC that in turn retained considerably more favourable alleles than standard GS. This allowed IND-RA and IND-HE, and to a lesser extent WGS, GOCS, and IND-OC, to increase the frequency of these favourable alleles to higher levels beyond cycle 10 as compared to GS, in a pattern that closely resembles genetic gain. For maintaining SNP alleles in general, which reflects a combination of what happens near QTL and at neutral loci, all methods show a very similar trend as for the favourable QTL alleles. As compared to IND-HE, IND-RA managed to retain slightly more alleles both for SNPs in general and favourable QTL alleles in particular.

When a stronger restriction on inbreeding is imposed for GOCS and IND-OC (figure 5.3; right panel) these methods retain slightly more favourable QTL alleles, accounting for a similar total effect and presumably thus with similar effect distribution, as compared to IND-HE and IND-RA. A parallel increase is observed for the number of maintained SNP alleles in general. Yet, these improvements are combined with a lower performance for increasing the favourable QTL allele frequencies, suggesting that IND-RA and IND-HE achieve a better balance between selection and avoiding drift as compared to the OC-based methods GOCS and IND-OC.

5.4 DISCUSSION

5.4.1 *Genomic OCS increases gain without full inbreeding control*

We used a simulation framework based on a recurrent selection plant breeding scheme to compare several long-term genomic selection strategies. In this breeding scheme GOCS outperforms GS

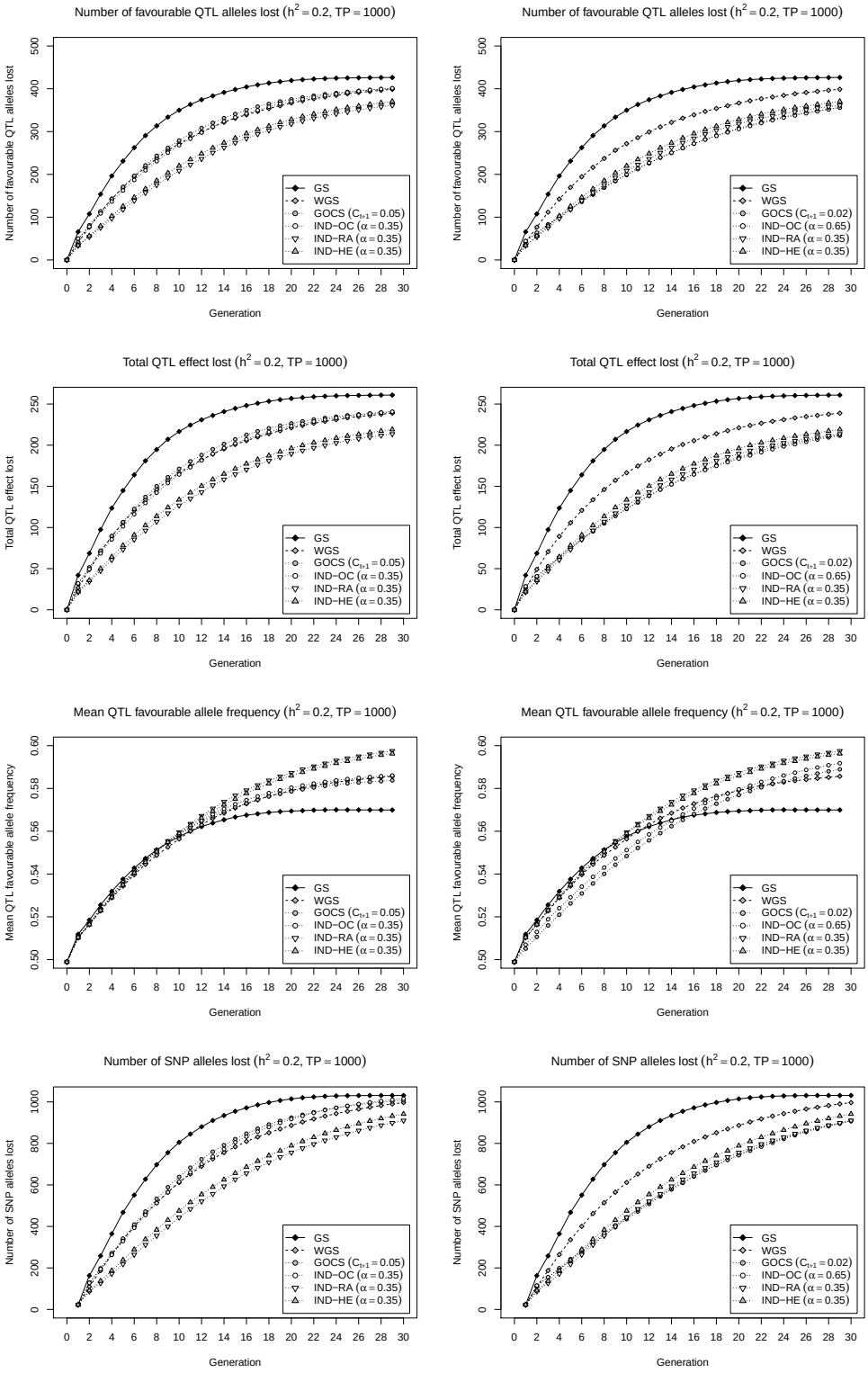


Figure 5.3: Number of favourable QTL alleles lost (first row), total QTL effect lost (second row), mean QTL favourable allele frequency (third row), and number of SNP alleles lost (fourth row) with the simulated selection strategies. Two scenarios were considered to set the parameters C_{t+1} and α: maintain the same short-term gain as WGS (left), or achieve a similar inbreeding rate ΔF_{IBS} (right). Results are reported for a low heritability ($h^2 = 0.2$) with a large initial training population (TP = 1000) and are averages of 200 simulation runs.

for long-term genetic gain and also results in a lower inbreeding rate, which is in line with earlier observations (Woolliams et al., 2015). Nearly identical results were obtained with the standard iterative algorithm (figure 5.1; right) and an index-based implementation of the same criterion (IND-OC; figure 5.2) which confirms that maximizing genetic gain while controlling inbreeding can also be achieved by optimizing an index that weighs both objectives, instead of maximizing gain while constraining inbreeding (Carvalheiro et al., 2010; Clark et al., 2013). The similar results of GOCS and IND-OC also suggest that in our special case, with a fixed number of selected individuals and equal contributions, the iterative heuristic of Meuwissen (2002) wrapped around OC's core optimization works well, as previously observed by Mullin and Belotti (2016).

Yet, despite that all simulations closely reach the imposed GOCS constraint $C_{t+1} = \Delta F_{target}$, the observed inbreeding rates ΔF_{IBS} and ΔF_{IBD} significantly deviate from this target level over generations. This differs from previous results by Sonesson et al. (2012) who did find that GOCS controls ΔF_{IBD} at the constraint value. The fact that the GOCS criterion $C_{t+1} = \mathbf{c_t}^\intercal \mathbf{G_t} \mathbf{c_t}/2$ does equal the inbreeding rate when no genomic selection is performed (figure A.2; left panel) suggests that the observed discrepancy is linked to selection. This hypothesis is confirmed by the theoretical expression of ΔF_{IBS} in terms of allele frequencies and their changes (see appendix A.2) showing that the inbreeding rate is the sum of $\mathbf{c_t}^\intercal \mathbf{G_t} \mathbf{c_t}/2$ and a second term involving cross products $\Delta_j(2p_j - 1)$ of frequencies and deltas. The latter term is expected to be close to zero when allele frequencies and their changes are independent such as in a scenario without selection. Under genomic selection, however, it becomes positive and increases over generations as selection pushes favourable alleles towards fixation. Once the frequency of a favourable allele exceeds 0.5, a beneficial Δ_j has the same sign as $2p_j - 1$, making their product positive. As such the inbreeding rate can be seen as the sum of a first component relating to the frequency changes at all loci in general, which is constrained by GOCS, and a second component that captures additional inbreeding due to specific selection pressure, which is not controlled by GOCS. The latter explains why the observed inbreeding rates generally exceed the target level when constraining $C_{t+1} = \mathbf{c_t}^\intercal \mathbf{G_t} \mathbf{c_t}/2$ to ΔF_{target}.

Knowing that GOCS does not fully control inbreeding under genomic selection still leaves the question why this was not observed by Sonesson et al. (2012) who only evaluated genomic IBD. Here the details of the simulation framework become important. We followed the same approach as Sonesson et al. (2012) to calculate genomic IBD, by working with a large collection of unique founder alleles positioned at equal distances across the genome. These founder al-

leles are specified irrespective of the QTL meaning that the same QTL allele can be linked to different founder alleles. As a result inbreeding at loci with favourable QTL can go undetected because the linked founder alleles are different, particularly when many founder alleles are still around in the population. This is likely the case in the simulations of Sonesson et al. (2012) where 200 individuals are selected in every generation and not in our setting, with only 20 individuals selected and therefore maximally 20 founder alleles per population except for the base population. In our simulations the effect of the large number of different founder alleles at the IBD markers is visible in the one-generation lag of ΔF_{IBD} as compared to ΔF_{IBS} (figure 5.1) and a less pronounced deviation of ΔF_{IBD} from the target inbreeding rate. Moreover, when selecting 50 instead of 20 individuals in every generation (figure A.2) ΔF_{IBD} starts below the target of 0.05—due to the large selection intensity it is impossible to achieve this fairly high inbreeding rate in early cycles—and then converges to the target level, as previously observed by Sonesson et al. (2012), while ΔF_{IBS} is still not controlled at the target level.

Along the lines we also conclude that ΔF_{IBS} measured with the actual marker panel will best represent loci under selection, while the multi-allelic IBD markers with unique founder alleles may better represent loci that are not under selection (although it is hard to imagine that these really exist with a 1000 QTL trait).

Taken together GOCS increases long-term genetic gain as compared to GS through better control of inbreeding, with the restriction of only fully controlling inbreeding at loci not subjected to selection.

5.4.2 *Weighted genomic selection*

The WGS method, which amplifies rare favourable allele effects when calculating GEBVs, also outperforms GS for genetic gain (figure 5.1; left panel) to a similar extent as in previous simulations by Jannink (2010)—thus confirming earlier results and validating our simulation framework. Compared to GS, WGS not only clearly controls the inbreeding rate but also yields very similar ΔF_{IBS} and ΔF_{IBD} (figure 5.1) suggesting that it operates equally on neutral loci and loci under selection, which is further supported by the parallel pattern for loss of favourable QTL alleles and SNPs (figure 5.3). This general effect on inbreeding is somewhat surprising because WGS only pushes frequencies of rare *favourable* alleles. Considering that the effect estimates for rare alleles in the genomic prediction model are likely very imprecise and might often even be of the wrong

sign, we can imagine that WGS in fact operates on rare alleles in general—at least in our setting with a 1000 QTL trait. It remains unclear, however, whether WGS would operate in a more trait-specific way when used for less complex traits. The intrinsic variability in the individual allele effect estimates might also be a reason for the somewhat unstable inbreeding control by WGS (figure 5.1) and makes this method highly dependent on external factors such as the trait heritability and training population size, suggesting that WGS may not always work as well as in our simulations.

Comparing WGS and GOCS we find that these methods provide the same short- and long-term genetic gain (figure 5.1) although it should be noted that both approaches can be further tweaked through parameter settings. Overall WGS appears a valid strategy to combine genetic gain with inbreeding control, and performs similarly as GOCS albeit with some hints of intrinsic instability.

5.4.3 *New index-based selection strategies*

We explored possible improvements on GOCS and WGS in a unified optimization framework with an objective that weighs genetic gain versus a diversity metric that is specific for the selection strategy. The optimal set of individuals to select is then approximated using the powerful parallel tempering local search metaheuristic, that maximizes the weighted index objective (see section 2.2.4).

A first strategy, IND-HE, aims to control inbreeding more generally than GOCS by also addressing the component specifically related to loci under selection. IND-HE does so by balancing genetic gain with high expected heterozygosity in the selected set. This is equivalent to minimizing the IBS-based inbreeding rate ΔF_{IBS} defined as relative decrease in expected heterozygosity.

In our simulation framework IND-HE clearly outperforms GOCS and IND-OC under various settings. In the scenario with similar short-term gain (figure 5.2; left panel) IND-HE realizes the same genetic gain during the first 10 cycles but with a lower inbreeding rate. This is likely because the IND-HE objective specifically quantifies inbreeding under selection, and as such captures the unavoidable penalty of fixing favourable alleles intrinsically linked to genetic gain. Therefore, to maintain gain, IND-HE leads towards selections where this inevitable additional inbreeding near QTL is compensated with lower inbreeding at neutral loci and at loci not yet under selection. As such, IND-HE is able to achieve the same genetic gain with less total inbreeding as compared to GOCS and IND-OC, due to which it retains more SNP alleles in general and

more favourable QTL alleles in particular (figure 5.3). The latter enables higher long-term gain in later cycles, when these conserved favourable alleles also start to move towards fixation. In conclusion, it seems that IND-HE provides a better balance between genetic gain and inbreeding control as compared to GOCS and IND-OC.

When keeping the realized inbreeding rates of GOCS and IND-OC below the level of IND-HE (figure 5.2; right panel) both methods retain a similar amount of favourable QTL alleles as IND-HE (figure 5.3). Yet, in this scenario GOCS and IND-OC show a considerably lower short-term gain than IND-HE without leading to a significant increase in long-term gain. We see two main reasons for this observation. First, to keep the realized inbreeding rate below that obtained with IND-HE, over all generations, we manually needed to correct for the additional inbreeding due to selection that is ignored by GOCS. As such, the target value for GOCS was considerably reduced from 0.05 to 0.02, resulting in a lower inbreeding rate than IND-HE up to cycle 10 (except from the first cycle) which impedes gain. In addition, GOCS limits squared allele frequency changes regardless of their direction, assuming that deviations from the current frequency always erode diversity due to inbreeding. The latter holds when pushing favourable allele frequencies already above 0.5 towards fixation. However, especially in early cycles, both gain and diversity could be simultaneously improved at certain loci, by amplifying favourable alleles with a frequency currently below 0.5. For such loci, the inbreeding term $\Delta_j(2p_j - 1)$ linked to selection is negative and may therefore compensate larger deltas at other loci. Yet, this term is ignored by GOCS due to which allele frequency changes may be overly constrained, potentially resulting in lower genetic gains. Also in this scenario we conclude that IND-HE is a more effective alternative for GOCS that provides a better inbreeding control, in particular also at loci under selection, and as such a better balance between genetic gain and diversity management.

The fact that the criterion used by GOCS and IND-OC ignores the direction of allele frequency changes is immediately visible in the first generation of our simulations, where IND-HE simultaneously realizes a strongly negative inbreeding rate and a high genetic gain (figure 5.2), which is similar to observations for WGS (figure 5.1) but outperforms GOCS and IND-OC (figure 5.2). We believe this is possible due to the presence of several large effect favourable alleles at very low frequencies in the population, in which case a strong selection for the trait can go hand in hand with increasing population-level heterozygosity. WGS and IND-HE are able to adapt to this situation and exploit this benefit while constraint-based methods like GOCS do not have the flexibility to go below the tar-

get inbreeding rate. This is a particular advantage for populations new to GS, as they are more likely to encounter positive large effect rare alleles as compared to populations already exposed to long-term GS. Even when relaxing the constraint to $\mathbf{c}_t^T \mathbf{G}_t \mathbf{c}_t / 2 \leqslant C_{t+1}$ instead of $\mathbf{c}_t^T \mathbf{G}_t \mathbf{c}_t / 2 = C_{t+1}$, following e. g. Pong-Wong and Woolliams (2007), GOCS would only go for an average squared frequency change below the target level if this yields higher immediate gain, which is unlikely, and still fails to recognize that amplifying rare favourable alleles is beneficial for both gain and diversity. The latter is inherent to the criterion used and thus also applies to IND-OC. Therefore, the start of a GS program may well be a point with a particularly pronounced difference between the OC-based methods and IND-HE because of the somewhat atypical situation with potentially many low frequency alleles, which may be addressed more effectively with a flexible index-based approach as compared to constraint-based strategies, and clearly requires methods that precisely model inbreeding also at loci under selection.

In an attempt to improve WGS we accommodated the specific focus on rare alleles in the diversity component of the index-based selection framework using the metric from Li et al. (2008), adjusted to operate on minor instead of favourable allele frequencies (IND-RA). The precise metric used is the mean of the log-transformed minor allele frequencies calculated in the set of selected individuals, with the log-transformation giving additional weight to rare alleles as compared to IND-HE. For genetic gain as well as for controlling the inbreeding rate IND-RA clearly outperforms WGS (figure 5.2).

We see two reasons for this observation. First, and likely most importantly, IND-RA resolves an intrinsic shortcoming of WGS, i. e. that truncation selection based on scores assigned to selection candidates cannot guarantee that the optimal set is selected. For example it is possible that multiple individuals carrying the same beneficial rare allele are selected while it might be better to choose complementary individuals that carry different rare alleles. The latter is favoured by IND-RA because the rare allele frequencies are evaluated at the level of the selected set. A second advantage is that IND-RA makes the management of rare alleles independent of the estimation of their effects in the genomic prediction model, that come with a high error and are dependent on external factors such as the trait heritability and training population size. This is likely the reason why IND-RA gives a more stable inbreeding rate than WGS (figures 5.1 and 5.2), and brings the additional benefit that rare alleles are managed in general and not specific for the trait under genomic selection and its genetic architecture. We conclude that IND-RA is superior to WGS for a selection strategy that avoids loss of rare alleles.

We do note that if desired it is possible to modify IND-RA to penalize loss of favourable alleles only, resulting in a more trait-specific approach, like WGS. However, experimenting with such alternatives (H De Beukelaer, G De Meyer; personal communication) gave similar or slightly worse gains suggesting that the simulated trait has too many underlying QTL to benefit from focusing on favourable alleles, due to inaccurate individual effect estimates.

Contrasting the IND-HE and IND-RA methods, that best represent their respective selection strategies, we find that both perform almost identically across a wide range of conditions (figures 5.2 and 5.3). This is not too surprising as avoiding loss of alleles is a specific aspect of controlling inbreeding. Both methods push towards high expected heterozygosity at population level and the main difference is their exponentially (IND-RA) versus quadratically (IND-HE) increasing focus on rare alleles. IND-RA indeed retains slightly more alleles than IND-HE (figure 5.3) but this is likely not important enough to affect long-term genetic gain within 30 cycles. Experiments with a different founder dataset (H De Beukelaer, G De Meyer; personal communication) having fewer QTL (100) and SNP markers (about 800) revealed that in such case there was a slight benefit of IND-RA over IND-HE when looking at long-term gain, likely because then 30 cycles were sufficient to realize some of the additional potential gained by retaining more favourable QTL alleles.

Overall, we conclude that IND-RA and IND-HE better balance genetic gain with avoiding loss of rare alleles, and controlling inbreeding in general, as compared to WGS and GOCS. Because our results indicate that general underlying mechanisms are at play, we believe that their relevance will extend from our simulation framework to many practical breeding settings.

5.4.4 *Practical considerations for implementing IND-RA and IND-HE*

Although the index-based optimization objectives for IND-RA and IND-HE are straightforward to compute for a given set of selection candidates, these methods need to identify the best subset from a huge number of possibilities. Achieving this within reasonable time requires intelligent combinatorial optimization algorithms, which is a major complication as compared to WGS where only wGEBV for all candidates have to be calculated, and as compared to GOCS where specific software is available. To allow high flexibility in terms of diversity metrics we used a general purpose metaheuristic to approximate the optimal selection, and chose for the par-

allel tempering algorithm because it is ideally suited for discrete optimization—as in the adopted breeding scheme where a fixed number of individuals are crossed with equal contribution. For breeding schemes involving unequal contributions other, continuous optimization algorithms might be preferred, such as differential evolution (Carvalheiro et al., 2010; Clark et al., 2013) or Lagrangian multipliers (Woolliams et al., 2015). In any case several practical tools are available for high-level heuristic optimization, including the JAMES framework described in chapter 3, that allow the user to specify any type of optimization objective that suits the breeding scheme at hand, and the type of diversity metrics deemed relevant.

We also note that the time needed for the search is not limiting—in our simulations it takes only a few seconds, and that would increase to at most a few minutes for e. g. multiple ten thousands of markers, because computation times increase linearly with the number of markers for the diversity measures used. Once the heuristic is in place it is straightforward to refine the objective function, for example by exploring other diversity metrics or including phenotypic diversity for relevant breeding objectives not yet under selection, or combinations of the above as the breeding program requires.

Furthermore, it should be mentioned that the log-criterion used by IND-RA could be further refined, for example by modifying the penalty assigned to losing an allele due to selection in equation (5.1). It is also possible to adjust the IND-RA and IND-HE methods for breeding schemes that allow unequal contributions by calculating the respective diversity metrics for the expected frequencies in the offspring, instead of the frequencies in the selection, and using a continuous optimization engine to find the contributions that best balance gain and diversity.

Finally, we note that in practice it may not be easy to find the best weight α, and that an index-based selection strategy does not allow to impose a predefined inbreeding rate. On the other hand Woolliams et al. (2015) argue that it is also not straightforward to set the target inbreeding rate in an OC-based approach, and we believe that simulating the breeding scheme at hand from its actual base population may be the most effective way to find appropriate parameter values, resulting in the desired balance between genetic gain and diversity, when using any selection strategy.

5.5 CONCLUSIONS

We investigated the performance of several long-term genomic selection strategies to balance genetic gain and population diversity,

in a simulation framework for a recurrent selection plant breeding scheme with equal contributions of a fixed number of individuals in each generation.

Genomic optimal contributions selection (GOCS), which extends pedigree-based OCS and constrains the realized genomic relationship among selected individuals, unexpectedly did not control the inbreeding rate at the target level. This happens because GOCS ignores the specific increase in inbreeding due to selection pressure at loci linked to QTL, which renders GOCS suboptimal for combining genetic gain with inbreeding control. We showed that this issue can be resolved with an index-based method (IND-HE) that balances genetic gain with expected heterozygosity in the selected set. IND-HE provides better results under a variety of settings, and is particularly more effective during the first cycles of a GS breeding program where gain and diversity may not yet be competing.

We also showed that weighted genomic selection (WGS), which amplifies rare allele effects when calculating GEBVs, is not fully effective at maintaining these rare alleles, or controlling inbreeding in general, because it was implemented as a truncation selection. An alternative method IND-RA that weighs genetic gain with rare allele frequencies in the set of selected individuals outperforms WGS with results that are very similar to IND-HE.

Both IND-HE and IND-RA provide a clearly better balance between genetic merit and diversity than GOCS or WGS and proved stable and effective irrespective of trait heritability and initial training population size. While requiring further testing in other breeding schemes, we believe that the inherent benefits of the IND-HE and IND-RA methods will transfer from our simulation framework to many practical breeding settings, and are a major step forward towards efficient long-term genomic selection.

ACKNOWLEDGEMENTS

We thank Jean-Luc Jannink for providing data, and advice on simulations and interpretation of the results.

DATA AVAILABILITY AND SOFTWARE

Raw data can be retrieved from http://www.hordeumtoolbox.org. Preprocessed data, the 200 pregenerated base populations used in the simulations, and all R and Java code are publicly available at https://github.com/hdbeukel/gs-simulation. Stochastic simu-

lations and analysis were performed in R 3.2.1 (R Core Team, 2015). Table A.1 provides a complete list of used R packages and their versions. We maximized the weighted indices of IND-OC, IND-HE and IND-RA in Java 8 using the JAMES framework (v1.2) presented in chapter 3. Java code was executed within R using the rJava package (Urbanek, 2015).

6

MARKER-ASSISTED GENE PYRAMIDING

SUMMARY

Some traits, including many disease and pest resistances, are controlled by a small number of genes. For such simple traits, breeders can target a fixed allele configuration at a small number of causal or linked loci. Efficiently obtaining this genetic ideotype from a given set of parental genotypes is known as the marker-assisted gene pyramiding problem. Existing methods either impose strong restrictions or use black box integer programming solutions, while we explore the power of an explicit heuristic approach that exploits the underlying genetic structure to prune the search space.

Gene Stacker combines an explicit directed acyclic graph model with a simple generation algorithm, providing both exact and heuristic pruning criteria to reduce the number of generated crossing schemes. We show that Gene Stacker yields good solutions for stacking problems of varying complexity. For more complex problems, the heuristics allow to obtain valuable approximations. Otherwise, fewer heuristics can be applied, resulting in an interesting quality-runtime tradeoff. Gene Stacker is competitive with previous methods and often finds better and/or additional solutions within reasonable time, because of the powerful heuristics.

The inherent flexibility of this approach allows to easily address important breeding constraints so that the obtained crossing schemes can be used in practice without major modifications. In addition, the ideas applied for Gene Stacker can be incorporated in and extended for a plant breeding context that for example also addresses complex quantitative traits or conservation of genetic background. The open-source Gene Stacker software is freely available at `http://genestacker.ugent.be`. The website also provides documentation and examples of how to use Gene Stacker.

6.1 INTRODUCTION

The breeding scheme simulated in chapter 5 was used to maximize yield, which is one of the most important but also most complex

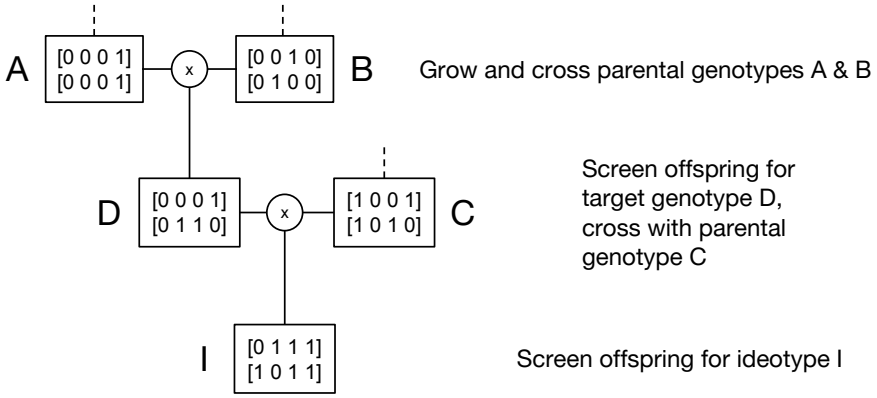


Figure 6.1: General crossing scheme layout (example). In each generation, a number of plants are grown and screened for the desired target genotypes. Crossings are then performed to provide new offspring to be grown, genotyped, and selected for use in the next generation. All crossings and selection targets are fixed in advance.

traits. Current marker-based plant breeding strategies designed to manage such complex traits focus on prediction, using hundreds or thousands of genome-wide markers, rather than on causality of individual markers. When breeding for simple traits however, such as many disease or pest resistances, we can go much further than just prediction of genetic value. Because such simple traits are controlled by a small number of genes we can precisely define the breeding target, looking at the few involved genes only, and predesign a detailed crossing scheme to efficiently obtain this so-called ideotype from the available resources.

In this chapter we develop an explicit framework to deal with such *foreground* markers. The objective is to design a crossing scheme that efficiently stacks a small number of favourable trait alleles (causal or tightly linked) present in a set of parental genotypes. This is known as the marker-assisted gene pyramiding or gene stacking problem. A crossing scheme consists of a number of generations in which plants are grown and screened to identify desired genotypes. These targets are selected for crossings, generating offspring to be grown, genotyped, and selected for use in the next generation, until the ideotype is obtained. An example with 3 parental genotypes is given in figure 6.1. Because the number of possible crossing schemes grows exponentially with the number of loci and parental genotypes, it is very challenging to design good schemes. With n loci, even a single crossing may produce a vast amount of up to $\mathcal{O}(4^n)$ possible offspring which are all candidates to be selected.

There are two main aspects that define a crossing scheme: the target genotypes aimed for in each generation (selection problem) and the crossings to be performed with these selected targets (scheduling problem). Important properties of a crossing scheme are the number of generations (time) and the size of the offspring (cost) that needs to be grown and screened to obtain the target genotypes in each generation. The latter is inversely proportional to the probability of observing these targets among the offspring.

Previous research on this topic has mainly focused on providing general guidelines for plant breeders (Ishii and Yonezawa, 2007; Ye and Smith, 2008) while only few papers offer a systematic algorithmic approach. Servin et al. (2004) considered restricted parental genotypes and represented crossing schemes as binary trees. For each crossing, the progeny that inherits all favourable alleles from both parents is selected, i. e. the selection problem is not optimized. An exhaustive algorithm is applied to generate all possible crossing schemes by iteratively combining smaller schemes through new crossings. Later, integer programming approaches were developed that use general purpose solvers like CPLEX to construct optimal schemes. Xu et al. (2011) perform a multi-objective optimization to fix desirable alleles while maintaining genetic variability at some remaining loci when possible. Only the selection problem is considered: each target allele in the ideotype is assigned an originating parental genotype and arbitrary minimum-depth binary trees are used to stack the genes according to this assignment. Canzar and El-Kebir (2011) provide a more powerful mixed integer programming (MIP) implementation were crossing schemes are modelled as directed acyclic graphs (DAGs) that allow reuse of material. Both the selection and scheduling problem are considered and the model incorporates a constraint on the number of offspring generated from one crossing, taking into account that the number of seeds produced per crossing strongly differs across plant species.

The CPLEX studio sold by IBM is a well-known tool for various types of mathematical optimization. Academics and students can use it for free. For more info, see `http://cplex.com`.

We introduce Gene Stacker, which combines an explicit DAG model, extending that of Canzar and El-Kebir (2011), with a pruned generation algorithm, inspired by the simple exhaustive search of Servin et al. (2004). We demonstrate that this works for small problems while more complex problems require supplementary heuristic pruning criteria that exploit the genetic structure to skip well-chosen parts of the search space. The proposed heuristics provide an interesting quality-runtime tradeoff. Gene stacker is not only a flexible and performant marker-assisted gene pyramiding tool with direct practical applications, but also a framework that can be extended, for example to also optimize for complex quantitative traits.

6.2 MATHEMATICAL MODELLING

6.2.1 *Encoding of genotypes*

A diploid phase-known genotype $G = (G_1, \ldots, G_k)$ consists of an ordered sequence of $k \geqslant 1$ chromosomes, each represented by a $2 \times n_i$ matrix G_i of alleles, where n_i is the number of considered loci on the i-th chromosome. The rows $G_{i,1}$ and $G_{i,2}$ of matrix G_i are called haplotypes and each correspond to one of the two homologous chromosomes of a diploid species. Note that interchanging the haplotypes (rows) of a chromosome $G_i \in G$ does not affect the genotype. The columns $G_i(j), j = 1, \ldots, n_i$, correspond to the considered loci in chromosome G_i and binary values (0/1) indicate the absence or presence of specific alleles. At every locus $0 \leqslant j \leqslant n_i - 1$ of chromosome G_i there are two alleles $G_{i,1}(j)$ and $G_{i,2}(j)$. This j-th locus is homozygous if $G_{i,1}(j) = G_{i,2}(j)$, else it is heterozygous. A genotype is said to be homozygous if all considered loci in each chromosome are homozygous.

In this context, calling a genotype "homozygous" indicates that all of the few considered loci are homozygous. This does not say anything about the entire genome and should for example not be confused with fully homozygous inbred lines.

6.2.2 *Recombination rates*

When crossing two diploid genotypes P and Q, each parent produces a haploid gamete and fusion of these gametes yields the diploid genotype of the child. A gamete $H = (H_1, \ldots, H_k)$ produced by genotype $P = (P_1, \ldots, P_k)$ consists of a series of haploid chromosomes H_i which each comprise a single haplotype and which are each (independently) obtained from the respective diploid chromosome P_i. A diploid chromosome can yield a number of different haplotypes due to recombination of alleles (crossover events). Given that we know the distance between any pair of loci on the same chromosome, we can convert these distances to crossover rates $r_{i,p,q}$ corresponding to the probability that a crossover will occur between loci p and q on the i-th chromosome, e. g. using the mapping function of Haldane (1919). Then, the probability $\Pr[P_i, Q_i \rightarrow G_i]$ that chromosomes P_i and Q_i will yield haplotypes which together form the new chromosome G_i is computed using formulas described by Canzar and El-Kebir (2011). For details, see appendix B.1. As Gene Stacker explicitly models multiple chromosomes, the final probability $\Pr[P, Q \rightarrow G]$ of producing the entire phase-known genotype G when crossing parents P and Q is

computed by multiplying the probabilities with which each chromosome is independently produced:

$$\Pr[P, Q \to G] = \prod_{i=1}^{k} \Pr[P_i, Q_i \to G_i].$$

6.2.3 *Population size*

Each genotype among the possible outcome of a crossing is a candidate to be selected in the next generation. However, such target genotype can only be selected if it actually occurs among the offspring. Thus, a sufficient amount of offspring should be generated so that the targets are expected to be produced. Consider a crossing of genotypes P and Q and a target genotype G that is produced with probability $\rho = \Pr[P, Q \to G]$. Given a desired success rate γ', the corresponding population size $N(\rho, \gamma')$ indicates the number of offspring that has to be generated so that the probability of obtaining at least one occurrence of G is at least γ' (Canzar and El-Kebir, 2011; Servin et al., 2004):

$$N(\rho, \gamma') = \begin{cases} \left\lceil \dfrac{\log(1-\gamma')}{\log(1-\rho)} \right\rceil & \text{if } \rho < 1 \\ 1 & \text{otherwise} \end{cases} \tag{6.1}$$

Gene Stacker ensures a global success rate γ (e. g. 95%) by setting a success rate $\gamma' = \sqrt[n]{\gamma}$ for each individual target, where n is the total number of targets obtained from crossings that can produce more than one possible child (i. e. crossings with uncertainty about the outcome). The total population size of a crossing scheme is equal to the sum of the population sizes required to obtain all target genotypes aimed for through the scheme and reflects the cost of the scheme. When several different genotypes or multiple occurrences of a specific genotype are targeted among offspring grown from the same seed lot, it is possible to compute a (lower) joint population size expressing the number of offspring that has to be generated to simultaneously obtain all targets (see appendix B.2).

6.2.4 *Extended DAG model*

Gene Stacker models a crossing scheme as a directed acyclic graph (DAG) with three types of nodes:

- *Seed lot nodes*: represent seeds obtained from a crossing, modelling the probability distribution of all phase-known geno-

types that may be produced. The source nodes of the graph are seed lot nodes from which the parental genotypes are grown. These initial seed lot nodes are assumed to be genetically uniform, i. e. they contain only one fixed phase-known genotype, and never to be depleted. Every internal seed lot node has a single crossing node as its parent. Edges leaving from a seed lot node are directed towards one or more plant nodes in any subsequent generation.

- *Plant nodes*: represent target genotypes to be selected from offspring grown from a specific seed lot. A plant node is labeled with its phase-known genotype and required population size (groups of plant nodes that are simultaneously obtained from the same seed lot are labeled with the required joint population size instead). If more than one occurrence of the respective genotype is targeted, the desired number of duplicates is indicated. Every plant node has a single seed lot node as its parent. Edges leaving from plant nodes lead to crossing nodes in the same generation.
- *Crossing nodes*: represent crossings with two plants from the same generation, resulting in a seed lot available in the next and all following generations. A crossing node is labeled with the number of times that the crossing is to be performed (if more than once). Every crossing node has two (not necessarily distinct) plant nodes as its parents. A single edge leaves from every crossing node to a seed lot node in the next generation.

Crossing a plant with itself is referred to as "selfing". The fact that this is possible for many plant species is one very important difference between animal and plant breeding.

Figure 6.2 shows a crossing scheme with 3 generations and a total population size of 1197. It is assumed here that every crossing provides about 250 seeds and that each plant can be crossed twice (or selfed once). Circular nodes represent seed lot nodes, rectangular nodes are plant nodes, and diamonds are crossings. Nodes which are aligned at the same vertical level are part of the same generation. The source nodes cover the 0-th generation, and each subsequent level of seed lot nodes starts the next generation. Gene Stacker's model allows reuse of plants (within a generation) as well as remaining seeds (across generations) and is an extension of the original DAG model from Canzar and El-Kebir (2011) which uses a single node type corresponding to Gene Stacker's plant nodes.

6.2.5 *Linkage phase ambiguity*

Gene Stacker is entirely based on phase-known genotypes as this allows to infer the distribution of possible offspring from a crossing. However, in practice, the linkage phase of a genotype is not directly

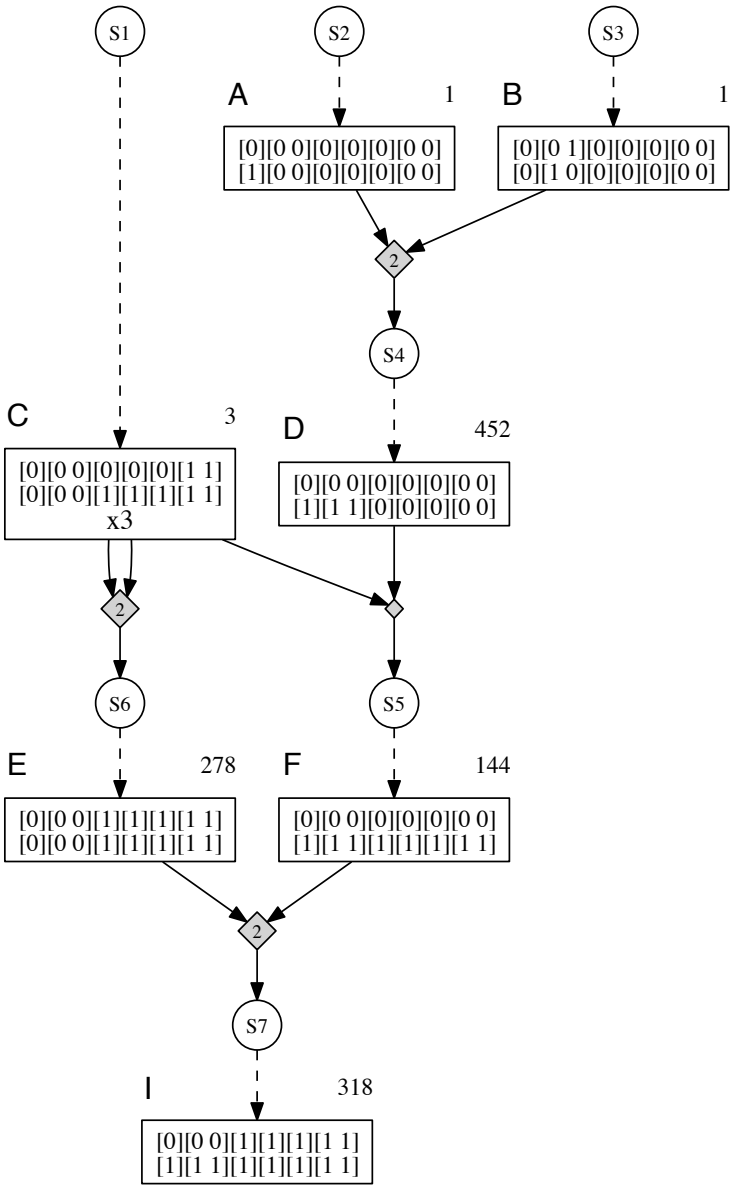


Figure 6.2: An example crossing scheme according to Gene Stacker's DAG model, with 3 generations and a total population size of 1197 (sum of population sizes required to obtain all target genotypes, as indicated at the corresponding plant nodes). It is assumed that every crossing yields about 250 seeds and that each plant can be crossed twice (or selfed once). First, parental genotypes A and B are crossed. This crossing is performed twice to provide a sufficient amount of seeds to obtain the target genotype D among the offspring. Subsequently, D is crossed with the third parental genotype C and the latter is also crossed with itself (twice). To be able to perform each of these crossings, 3 duplicates of C are grown. Finally, E and F are crossed to produce the ideotype I.

observed (Browning and Browning, 2011). Therefore, it is important to monitor the linkage phase ambiguity (LPA) which expresses the probability that a genotype will have an undesired linkage phase. The observed allele frequencies of a genotype G are referred to as $\widetilde{G}$. When crossing genotypes P and Q, the probability $Pr[P, Q \to \widetilde{G}]$ of obtaining any genotype with the same allele frequencies as G is computed as follows:

$$Pr[P, Q \to \widetilde{G}] = \sum_{G', \widetilde{G}' = \widetilde{G}} Pr[P, Q \to G'].$$

Then, the linkage phase ambiguity of G is equal to

$$LPA[P, Q \to G] = 1 - \frac{Pr[P, Q \to G]}{Pr[P, Q \to \widetilde{G}]}.$$

For target genotypes with non-zero linkage phase ambiguity, the inferred LPA is included in the label of the corresponding plant node. The overall LPA of a crossing scheme is defined as the probability that at least one target genotype aimed for through the scheme will have an undesired linkage phase, and can easily be computed from the individual, independent ambiguities.

6.3 OPTIMIZATION STRATEGY

6.3.1 *Approximated Pareto front*

Gene Stacker approximates the Pareto front of crossing schemes with a minimum number of generations, total population size, and overall linkage phase ambiguity, possibly subject to a number of crop specific and practical constraints. Upper limits can be set for (a) the number of generations (required); (b) the overall linkage phase ambiguity; (c) the total number of crossings; (d) the population size per generation; and (e) the number of crossings with each plant. Also, the expected number of seeds obtained from a crossing can be specified. As described in section 2.3, the Pareto front contains all solutions within the constraints that are not dominated by any other valid solution, where a scheme S' dominates another scheme S if it is at least as good for every objective and better for at least one objective. All non-dominated schemes are optimal in some sense as they provide tradeoffs with respect to the different objectives. The Pareto front approximation constructed by Gene Stacker contains all obtained schemes for which no dominating other solution has been found.

6.3.2 *Optimization algorithm*

The main generation algorithm used by Gene Stacker is a breadth-first search (see section 2.2.1) similar to the exhaustive strategy from Servin et al. (2004), where exact and heuristic pruning criteria are applied to reduce the number of generated schemes. The search space is traversed as a tree by starting with the smallest possible schemes, i. e. those which simply grow one of the parental genotypes, and iteratively extending schemes through additional crossings. There are two types of extensions: (a) selfing the final plant of a scheme; or (b) combining two schemes through a crossing of the final plants of both schemes. Every phase-known genotype among the offspring is then considered as a selection target, yielding a (possibly huge) number of extended schemes. With each node in the search tree corresponds a partial crossing scheme that is extended in all possible ways, either through a selfing or by combining it with any of the previously obtained partial schemes, i. e. those found at the nodes that were visited so far. This means that the number of branches per node rapidly increases while traversing the tree and that each generated partial scheme needs to be stored so that we can later combine it with other schemes. The generation strategy thus has an inevitable high memory pressure and intuitively corresponds to a breadth-first search approach.

When combining two schemes, their generations can be aligned in different ways. Plant or seed lot nodes occurring in both schemes which are aligned in the same generation of the combined scheme are dynamically reused. Gene Stacker greedily discards any alignments that are not Pareto optimal. Therefore, the main algorithm is already not entirely exact. However, the impact of this greedy approach on the solution quality is expected to be small—it mainly prevents the introduction of most likely redundant generations and favours alignments with the highest amount of reuse leading to a reduced total cost.

Figure 6.3 shows two equally good alternatives of the same scheme, i. e. with the same number of generations, total population size, and linkage phase ambiguity, that were obtained from a different alignment of the generations of the two combined smaller schemes. Both alignments are retained and will be considered for further extension. Different alignments may also provide tradeoffs between objectives, as shown in figures 6.4 and 6.5. Such Pareto optimal alignments are again each queued for further extension. In contrast, figures 6.6 and 6.7 both show two alternative alignments where one is greedily discarded (right) because it is dominated by the other

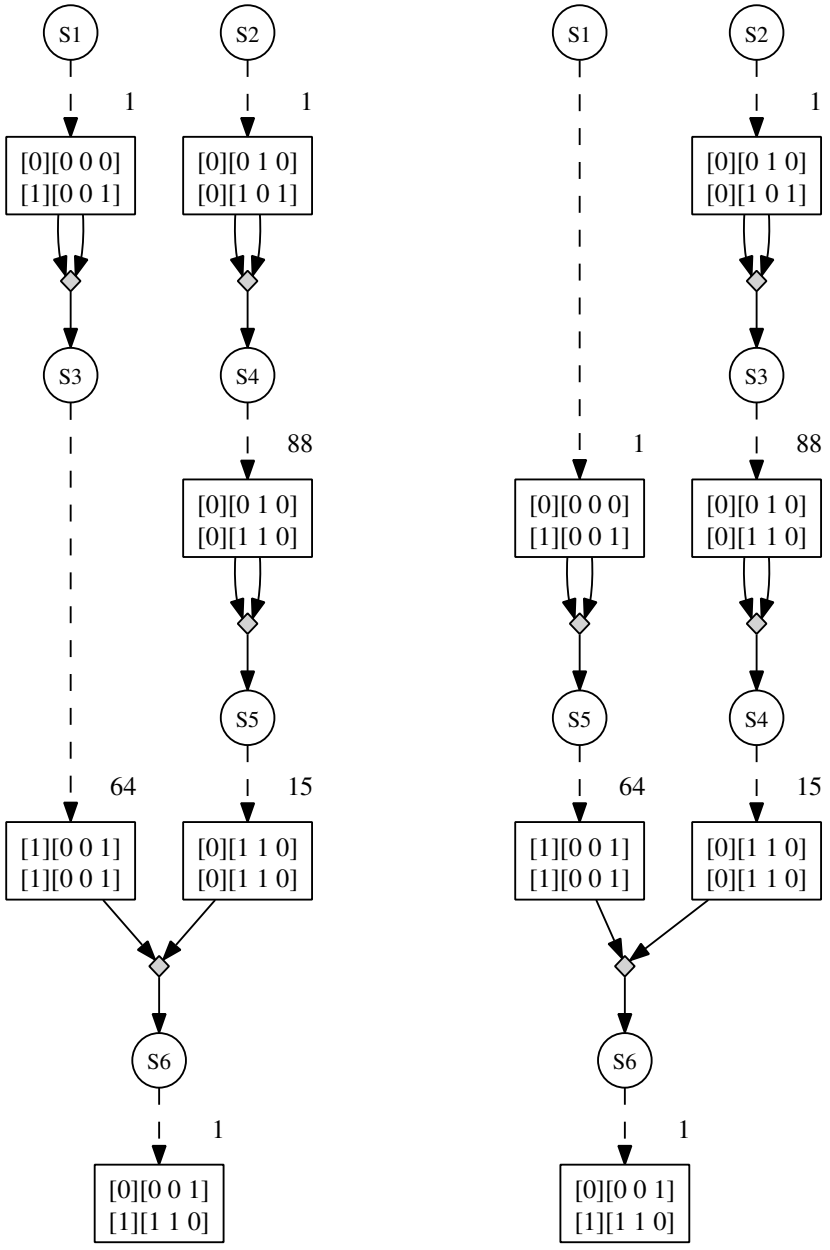


Figure 6.3: Alternatives of the same scheme, obtained from a different alignment of generations when combining two smaller schemes. Both alternatives are equally good in terms of all three considered objectives (number of generations, total population size, and overall linkage phase ambiguity). Both alternatives are retained and will be considered for further extension.

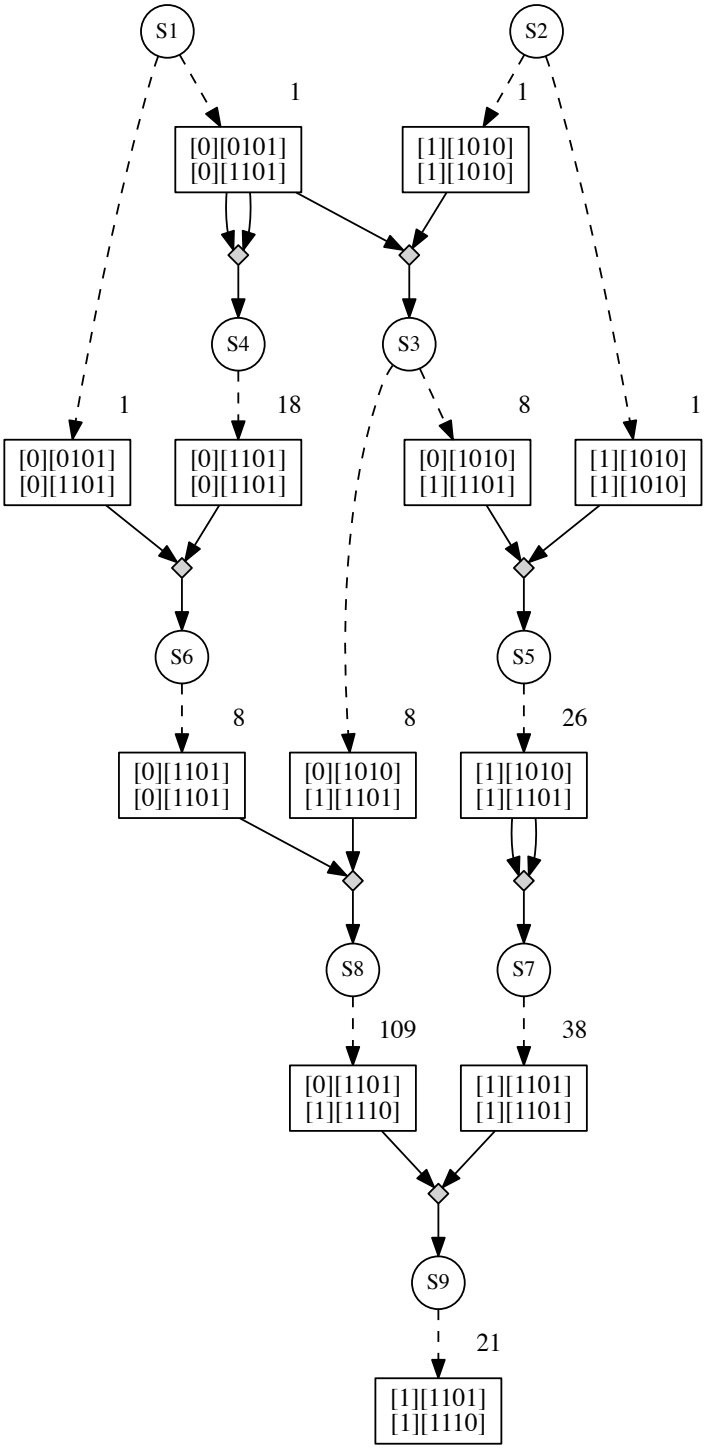


Figure 6.4: Alternative alignment of generations for the crossing scheme shown in figure 6.5. This option has one less generation but grows the same plant twice from seed lot S3, in two subsequent generations, which requires a repeated screening and therefore slightly increases the total cost (population size).

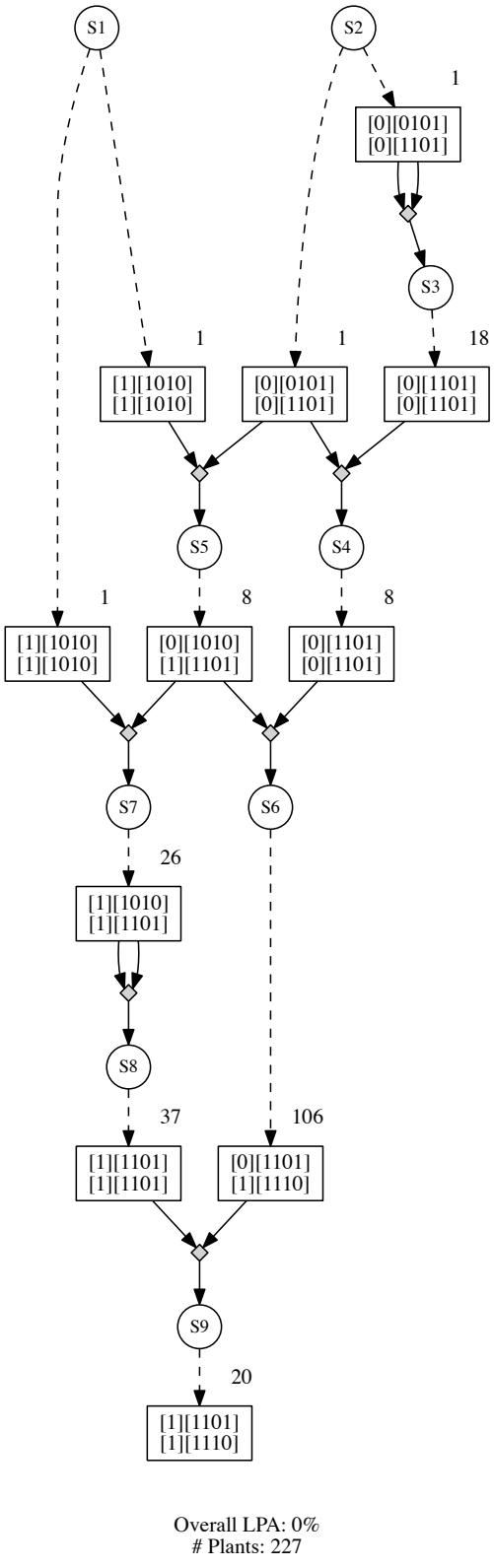


Figure 6.5: Alternative alignment of generations for the crossing scheme shown in figure 6.4. This option reuses the plant grown from seed lot S5—which corresponds to seed lot S3 in figure 6.4—to perform two crossings in the same generation. As a result, the total population size is decreased but this comes at the cost of an additional generation. Therefore, both Pareto optimal alignments are retained and considered for further extension.

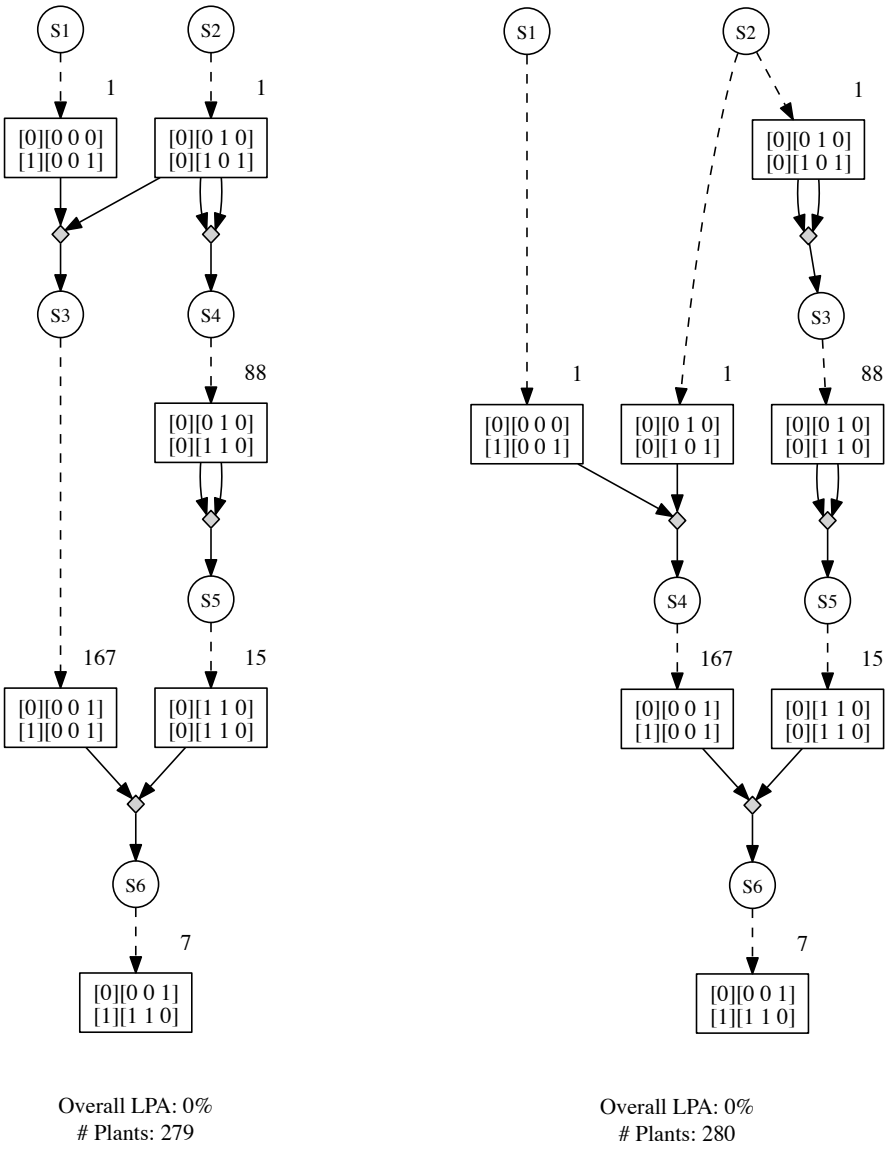


Figure 6.6: This figure shows two alternative alignments of generations in a crossing scheme, where the left alignment is preferred over the right alignment since it has a lower total population size, the same number of generations, and the same overall linkage phase ambiguity. By already crossing the two initial parents in the first generation (left) one of these can be reused for a simultaneous selfing, while this plant has to be regrown if the former crossing is postponed to the second generation (right). Reuse of material is often very beneficial, especially when it has been obtained at a high cost. The left alignment is retained, but the right alignment is greedily discarded.

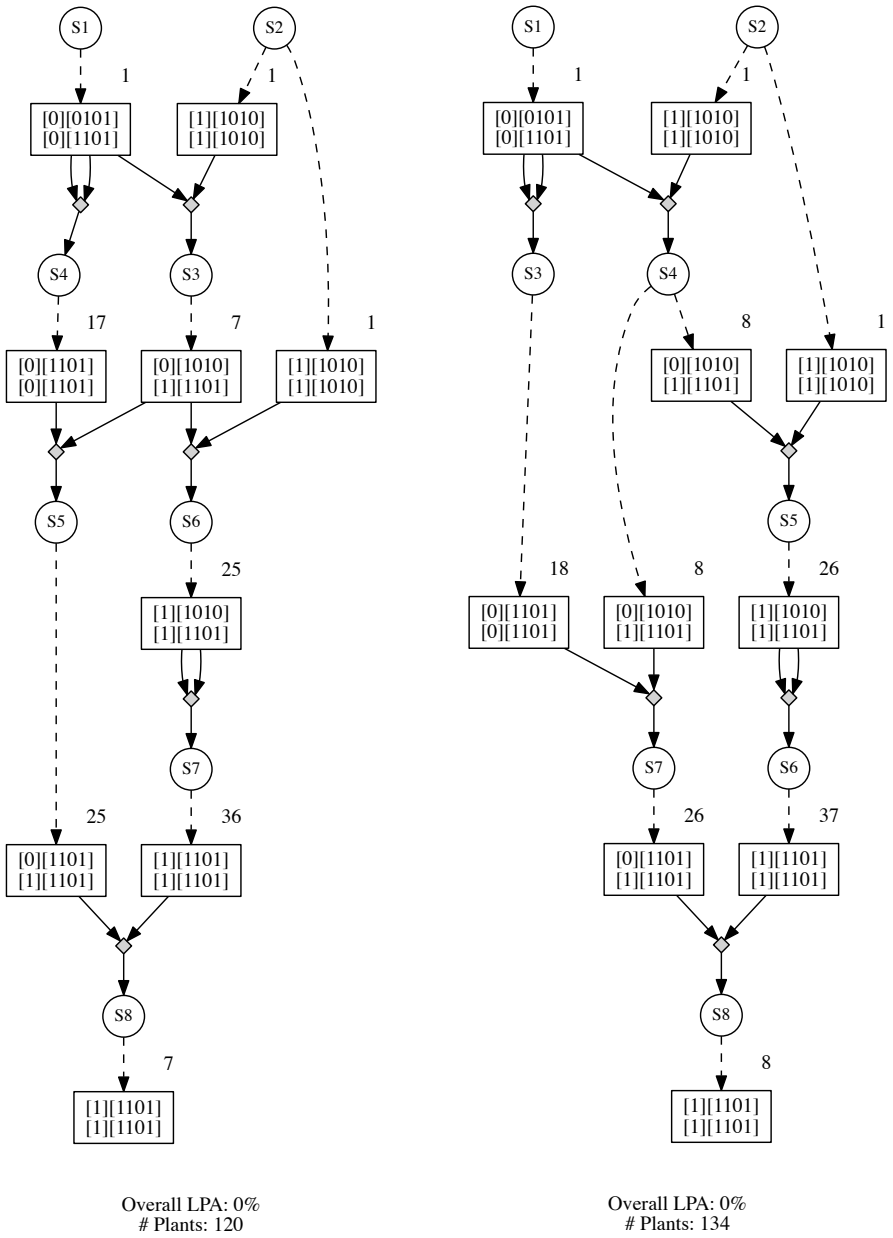


Figure 6.7: This figure provides a second, more complex example of two alternative generation alignments where, again, the left alignment reduces the total population size through reuse of plants, now with about 10%, without affecting the number of generations. Therefore, only the left alignment is retained, and the other option is greedily discarded by Gene Stacker.

alignment (left) which has a smaller population size, and the same number of generations and linkage phase ambiguity.

If an extension yields a new scheme in which the ideotype is obtained, the Pareto front is updated accordingly. Else, the scheme is queued for further extension, unless it is predicted that every completed extension will either be dominated by an already obtained solution or violate the constraints. Such pruning reduces the number of constructed schemes and therefore the runtime and memory footprint of the algorithm. Gene Stacker includes several heuristics that further reduce the search space by exploiting the underlying genetic structure to skip non-promising branches of the search tree. Well-designed heuristics may result in large speedups with only a slightly higher probability of obtaining suboptimal solutions, which are often still close to the optimum. The search terminates when there are no more schemes to be further extended.

Algorithm 6.1 provides the main pseudocode of the Gene Stacker algorithm. Given a set of parental genotypes $\mathcal{G}$ and the desired ideotype $\mathcal{I}$, Gene Stacker approximates the Pareto front $\mathcal{F}$ containing several high-quality schemes that provide tradeoffs in terms of the different objectives. The queue $\mathcal{Q}$ contains those schemes that still have to be extended and the algorithm iteratively dequeues partial schemes S from $\mathcal{Q}$ to create larger schemes S_{new} by (a) selfing the final plant of S; and (b) combining S with each previously extended scheme S' through a crossing of the final plants of both schemes. Every element $C \in \mathcal{Q}$ consists of a series of $a \geqslant 1$ alternatives $S[0], \ldots, S[a-1]$ of the same scheme. These alternatives arise because, as described above, there are several ways to align the generations of two smaller schemes S and S' when combining them into a larger scheme S_{new}. Each generation of S_{new} contains either a single generation from S or S', or consists of the alignment of two generations; one from each of the smaller schemes.

Whenever a crossing or selfing is performed to extend a scheme, the corresponding seed lot $\mathcal{S}$ is constructed by (a) inferring all possible haplotypes that can be produced from each chromosome of both parents; (b) creating all pairwise combinations, per chromosome, of haplotypes produced by both parents; and (c) making all combinations of the obtained chromosomes. This yields the set of possible offspring. During generation, the corresponding probabilities and linkage phase ambiguities are computed.

In case the final plant of a partial scheme S has been selfed, Gene Stacker considers each genotype G in the constructed seed lot $\mathcal{S}$ to be fixed as a possible selection target. For each genotype G, the alternatives of a larger scheme S_{new} are created by attaching the

Algorithm 6.1 Pseudocode of the generation algorithm used by Gene Stacker.

```
function GeneStacker(𝒢, ℐ)
    𝒬 ← [ ]                                          ▷ queue of schemes to be extended
    𝒫 ← [ ]                                          ▷ previously extended schemes
    ℱ ← [ ]                                          ▷ current Pareto front approximation
    for all parental genotypes P ∈ 𝒢 do
        add minimal scheme growing P to 𝒬            ▷ queue all minimal schemes for extension
    end for
    while 𝒬 not empty do
        S ← dequeue element from 𝒬                   ▷ scheme to be extended (⩾ 1 alternatives S[i])
        𝒮 ← Self(final plant of S)                   ▷ compute seed lot obtained by selfing
        for all genotypes G ∈ 𝒮 do                   ▷ consider each genotype in 𝒮 as next target
            S_new ← [ ]
            for i = 0, ..., |S| − 1 do               ▷ consider all alternatives of S
                S_new[i] ← attach selfing, 𝒮 and G to S[i]   ▷ extend S[i] to create S_new[i]
            end for
            RegisterScheme(S_new, ℐ, ℱ, 𝒬)           ▷ register new scheme (all alternatives)
        end for
        for all S′ ∈ 𝒫 do                            ▷ combine with previous schemes
            A ← [ ]
            for i = 0, ..., |S| − 1 do               ▷ align alternatives of S and S′ (pairwise)
                for j = 0, ..., |S′| − 1 do
                    B ← Align(S[i], S′[j])           ▷ construct all alignments of S[i] and S′[j]
                    Add all B′ ∈ B to A              ▷ store constructed alignments
                end for
            end for
            FilterAlignments(A)                      ▷ remove suboptimal alignments
            𝒮 ← Cross(final plant of S, final plant of S′)   ▷ compute seed lot obtained from crossing
            for all genotypes G ∈ 𝒮 do               ▷ consider each genotype in 𝒮 as next target
                S_new ← [ ]
                for i = 0, ..., |A| − 1 do           ▷ consider all retained alignments
                    S_new[i] ← attach G to A[i]      ▷ complete A[i] to create S_new[i]
                end for
                RegisterScheme(S_new, ℐ, ℱ, 𝒬)       ▷ register new scheme (all alternatives)
            end for
        end for
        Add S to 𝒫                                   ▷ add S to list of already extended schemes
    end while
    return ℱ                                         ▷ return final Pareto front approximation
end function

function RegisterScheme(S_new, ℐ, ℱ, 𝒬)
    for i = 0, ..., |S_new| − 1 do                   ▷ consider all alternatives of S_new
        ResolveDepletedSeedLots(S_new[i])            ▷ resolve any depleted seed lots
        if ideotype ℐ obtained and constraints satisfied then
            Update ℱ with new solution S_new[i]      ▷ update Pareto front
            Remove S_new[i] from S_new               ▷ discard alternative (complete)
        else if Prune(S_new) then                    ▷ check (heuristic) pruning criteria
            Remove S_new[i] from S_new               ▷ discard alternative (pruned)
        end if
    end for
    if |S_new| > 0 then                              ▷ check if any alternatives remain
        Add S_new to 𝒬                               ▷ queue scheme for further extension
    end if
end function
```

performed selfing, the obtained seed lot $\mathcal{S}$ and the targeted genotype G to each alternative $S[i]$ of S.

When combining two partial schemes S and S' through a crossing of their final plants, Gene Stacker first creates all alignments A of all pairs of alternatives $S[i]$ and $S'[j]$. Alignments are constructed *bottom-up*: first, the new crossing node is created, joining the crossed plants, and then the alignments are further completed by repeatedly inserting the previous generation from either $S[i]$, $S'[j]$, or both. Plant nodes and seed lot nodes occurring in both smaller schemes, which end up being aligned in the same generation of the new scheme, are dynamically reused. Of all constructed alignments within the constraints, only Pareto optimal ones are retained. Other alignments are greedily discarded. Then, for every genotype G in $\mathcal{S}$, the alternatives of a larger scheme S_{new} are created by attaching G as the next target to each retained alignment $A[i]$.

For each alternative of every newly created scheme S_{new} it is checked whether there are any depleted seed lots, i. e. seed lots from which more seeds are taken than the amount provided by the performed crossing(s). In such case, Gene Stacker indicates that the crossing should be performed multiple times to provide additional seeds. For this, it may be necessary to have several duplicates of the crossed genotypes, taking into account the number of crossings that can be performed with a single plant. This affects the population sizes and may introduce new depleted seed lots. Therefore, this process is iterated until all depleted seed lots have been resolved.

When the ideotype $\mathcal{I}$ is obtained, each alternative $S_{new}[i]$ for which all constraints are satisfied is registered in the Pareto front $\mathcal{F}$. If $S_{new}[i]$ is not dominated by any other solution currently contained in $\mathcal{F}$, it is added to $\mathcal{F}$ and all schemes dominated by $S_{new}[i]$ are removed from $\mathcal{F}$. If the ideotype is not yet obtained, S_{new} is added to the queue $\mathcal{Q}$ for further extension, where some alternatives $S_{new}[i]$ may be discarded by one of the applied heuristics, or if it is predicted that all extensions will violate the constraints or will be dominated by an already obtained solution (pruning). Note that in the actual implementation, some pruning criteria are checked at other points in the code to enable early pruning, for example when computing the seed lot obtained from a crossing, when combining two specific partial schemes, or when considering to fix a specific genotype G as the next selection target. These technical implementation details do not modify the general search strategy and are best explained by looking at the source code, available from `http://genestacker.ugent.be`.

Gene Stacker continues until the queue $\mathcal{Q}$ is empty. Termination is guaranteed because of a required constraint on the number of

generations, which will eventually always be violated meaning that no new schemes are added to the queue.

6.3.3 *Exact pruning criteria*

Because the number of generations, total population size and overall linkage phase ambiguity are monotonically increasing, any partial scheme which is dominated by a previously obtained solution or which already violates the corresponding constraints may be discarded. In addition, some basic bounds are applied. For example, when combining two partial schemes, it is predicted whether this may yield a valid improvement over the current Pareto front approximation by inferring the minimum combined population size and linkage phase ambiguity from the set of non-overlapping plant nodes and seed lot nodes occurring in both schemes. Also, the minimum increase in population size and ambiguity caused by targeting any genotype among the offspring of the performed crossing is taken into account. Although these are local bounds that predict the impact of a single extension, they often cause significant speedups as creating all extensions of a given scheme is a very time consuming and memory intensive process.

Constructed seed lots are filtered based on the constraints. Genotypes with higher linkage phase ambiguity than the maximum allowed overall ambiguity are removed. Also, if at most m plants per generation are allowed, a genotype G obtained from crossing P and Q is discarded if

$$\Pr[P, Q \to G] < 1 - (1 - \gamma')^{\frac{1}{m}}.$$

Given that at most g generations are allowed, Gene Stacker prunes a significant number of branches when creating schemes with $g-1$ or g generations. At generation $g-1$ only genotypes from which the ideotype can be obtained through a single crossing are considered as possible selection targets, i. e. genotypes that can produce one of both desired haplotypes for every chromosome of the ideotype. Furthermore, in this penultimate generation, only those crossings which can produce the complete ideotype are performed. Obviously, in the final generation g, only the ideotype itself is considered as a target. These pruning criteria are very effective and yield huge speedups in the last two levels of the search tree.

6.4 HEURISTICS

In order to further reduce the search space we propose several heuristics that exploit the underlying genetic structure of the gene stacking problem. These heuristics can be switched on or off to control the balance between execution time and solution quality.

6.4.1 *Improvement-based heuristics*

Several of the provided heuristics are based on improvement of phase-known genotypes towards the ideotype. Improvement is expressed within a chromosome and a genotype is considered to be an improvement if at least one chromosome has improved. Gene Stacker uses two different improvement criteria: *weak* and *strong* improvement. First, the definitions of desired stretches and alleles are introduced.

Definition 6.1 (desired stretch). Given a chromosome C with k loci, take any of both haplotypes H of C. Then, the stretch $S^H_{i,j}$, $0 \leqslant i \leqslant j \leqslant k-1$, is defined as the part of H comprising the consecutive alleles at loci $i, i+1, \ldots, j$. The length of the stretch is denoted as $|S^H_{i,j}| = j - i + 1$. A stretch $S^H_{i,j}$ is desired if the respective chromosome of the ideotype contains a haplotype H' for which $\forall l, i \leqslant l \leqslant j, H(l) = H'(l)$.

Definition 6.2 (desired allele). A desired stretch of length one is also simply referred to as a desired allele. Informally, an allele is desired at a certain locus of a chromosome if that allele occurs at that locus in the respective chromosome of the ideotype.

Example 6.1. Given an ideotype chromosome

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

only the 1-allele is desired at the first locus, while both alleles are desired at the second and third locus. The chromosome

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

thus still lacks one desired allele, i. e. the 1-allele at the second locus, and also contains an undesired 0-allele at the first locus. The top haplotype $H = [0\ 0\ 0]$ contains two desired stretches of length one, corresponding to the desired alleles at the last two loci, which together form a desired stretch $S^H_{1,2} = [0\ 0]$ of length two. Similarly,

all three alleles contained in the bottom haplotype $H = [1\ 0\ 1]$ are desired. In addition, the larger stretch $S^H_{0,1} = [1\ 0]$ spanning the first two loci is also desired, but not the full haplotype.

As a second example, take the chromosome

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

which already contains all desired alleles at all loci. However, not all desired stretches have yet been obtained, because a different linkage phase is desired. △

The definition of weak improvement then follows.

Definition 6.3 (weak improvement). Given two chromosomes C, C' and the ideotype $\mathcal{I}$, C is a weak improvement over C' towards $\mathcal{I}$, denoted as $C \succ^{\mathcal{I}}_{w} C'$, if either (a) one of both haplotypes H of C contains a desired stretch $S^H_{i,j}$ which is not present in any of both haplotypes of C'; or (b) C homozygously contains a desired allele which does not occur in C' in homozygous state.

The first case favours the introduction of new or extended desired stretches and the second case rewards *stabilization* of desired alleles to prevent them from being lost during subsequent crossings.

Example 6.2. Take three chromosomes

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad C^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad C^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

and a respective ideotype chromosome

$$I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Then C^1 and C^2 are both weak improvements over C towards the ideotype, because C^1 contains a desired 1-allele at the first locus not found in C, and because C^2 stabilizes the already present desired 1-allele at the third locus, respectively. For the same reasons, C^1 and C^2 are also both weak improvements over each other. △

In addition to weak improvement, Gene Stacker also uses the following concept of strong improvement.

Definition 6.4 (strong improvement). Given a chromosome C, define M as the set containing all desired stretches $S^H_{i,j}$ occurring in any haplotype H that can be produced from C with at most one crossover. Then define the tuple (l_C, p_C) as

$$l_C = \max\{|S^H_{i,j}|; S^H_{i,j} \in M\}$$

and

$$p_C = \max\{\Pr[C \to S^H_{i,j}]; S^H_{i,j} \in M \,\&\, |S^H_{i,j}| = l_C\}$$

where $\Pr[C \to S^H_{i,j}]$ is the probability that C will produce any haplotype containing the stretch $S^H_{i,j}$. Now, take two chromosomes C and C', and an ideotype $\mathcal{I}$. Then C is a strong improvement over C' towards $\mathcal{I}$, denoted as $C \succ^{\mathcal{I}}_s C'$, if

$$(l_C > l_{C'}) \vee (l_C = l_{C'} \wedge p_C > p_{C'}).$$

To detect strong improvement, chromosomes are first compared based on the length of the longest desired stretch that may be produced with at most one crossover—an idea which has been previously proposed by El-Kebir et al. (2009). In case of equal lengths, the highest probability with which any such maximal desired stretch is produced from the two chromosomes, respectively, is compared.

Example 6.3. Take the same three chromosomes

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad C^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad C^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

and respective ideotype chromosome

$$I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

as in the previous example. While C^1 is a weak improvement over C, it is not a strong improvement because the newly introduced desired 1-allele at the first locus is separated from the already present desired stretch of 1-alleles at the last two loci. Therefore, neither the length of the maximal desired stretch that can be obtained with at most one crossover, nor the associated probability has increased. In contrast, C^2 is not only a weak but also a strong improvement over C because stabilizing the 1-allele at the third locus increases the probability of producing a gamete that contains the desired stretch of 1-alleles found at the last two loci. For the same reason, C^2 is a strong improvement over C^1.

Unlike C^1, both chromosomes

$$C^3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad C^4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

are strong improvements over C, as well as over C^1 and C^2, as the length of the maximal desired stretch that can be produced with at most one crossover has increased from two to three. △

All strong improvements are also weak improvements, but not vice versa. Thus, using strong improvement as a condition in the heuristics introduced below will prune more branches of the search tree. The rationale behind the definition of strong improvement is that selection targets requiring multiple crossovers are usually to rare to be considered, i. e. selecting them would require a too large population size. Therefore, introducing new isolated desired alleles or stretches, as allowed by the definition of weak improvement, is assumed not to be very beneficial if they can not be joined with a single crossover. Also, strong improvement is more demanding in terms of monotonicity. For weak improvement it is sufficient that a new desired allele is introduced somewhere in the genome, and it is allowed that previously obtained desired stretches are lost in the process. In contrast, strong improvement requires that continuously growing desired stretches are obtained. Again, stabilization of desired alleles or stretches into a homozygous state is rewarded by comparing based on the probability with which the maximal desired stretches are produced, when they have equal lengths.

Gene Stacker includes three heuristics which are based on improvement of genotypes towards the ideotype. The first heuristic (H0) is applied once to filter the parental genotypes $\mathcal{G}$.

Heuristic H0 (parental genotype filter). Discard each parental genotype $G \in \mathcal{G}$ for which $\exists G' \in \mathcal{G}, G' \neq G$, where G' is a strict weak improvement over G, i. e. $G' \succ_w^{\mathcal{J}} G \wedge \neg(G \succ_w^{\mathcal{J}} G')$.

The other heuristics are repeatedly applied to prune non-promising branches of the search tree.

Heuristic H1 (improvement over ancestors). Each genotype G is required to be an improvement over all ancestors, i. e. $G \succ_{\ldots}^{\mathcal{J}} A$ for each genotype A occurring on any path from a source node to G. It is also allowed that $G = A$ if G has a smaller linkage phase ambiguity, or occurs with a higher probability than A among the respective seed lot. The applied improvement criterion $\succ_{\ldots}^{\mathcal{J}}$ can be either weak (H1a) or strong improvement (H1b) .

Heuristic H2 (seed lot filter). When crossing genotypes P and Q, discard any genotype G from the obtained seed lot $\mathcal{S}$ for which $\exists G' \in \mathcal{S},\ G' \neq G$, with

$$G' \succ_{\ldots}^{\mathcal{J}} G \wedge \neg(G \succ_{\ldots}^{\mathcal{J}} G')$$

and both

$$\begin{aligned} Pr[P, Q \to G'] &\geqslant Pr[P, Q \to G] \\ LPA[P, Q \to G'] &\leqslant LPA[P, Q \to G]. \end{aligned}$$

Again, the applied improvement criterion $\succ_{\ldots}^{\mathcal{J}}$ can be either weak (H2a) or strong improvement (H2b) .

Heuristic H2 removes genotypes from $\mathcal{S}$ if a strictly better genotype is also available which requires equal or less effort to be obtained from $\mathcal{S}$, in terms of population size (probability) and linkage phase ambiguity.

6.4.2 *Optimal subschemes*

The following heuristic (H3) assumes that an optimal scheme consists of optimal subschemes.

Heuristic H3 (optimal subschemes). A distinct Pareto front $\mathcal{F}(G)$ is maintained for each genotype G, consisting of schemes with final genotype G. Such scheme S is only queued for further extension if it is not dominated by a previous scheme $S' \in \mathcal{F}(G)$. Moreover, extensions are only constructed if S is still contained in $\mathcal{F}(G)$ when it is dequeued. As an exception, selfing a homozygous genotype is always allowed.

The exception made in heuristic H3 allows efficient reuse of homozygous genotypes across generations with only a small increase in the number of explored branches of the search tree. Experiments showed that applying this heuristic generally results in very large speedups, but regularly also yields worse Pareto front approximations because the assumption that optimal schemes consist of optimal subschemes does not hold when reusing material. Therefore, we designed two dual run strategies where H3 is enabled in the first run only. The second run then benefits from the availability of an initial Pareto front approximation, which for example allows earlier pruning. Heuristic H3s1 follows this basic dual run strategy. Heuristic H3s2 also applies an additional seed lot filter in the second run that restricts the possible haplotypes for each chromosome to those occurring in a solution found in the first run. The overhead of the first run is usually much smaller than the speedup obtained in the second run, due to the availability of an initial Pareto front estimation, and the additional filter in case of H3s2.

6.4.3 *Pareto optimal seed lot*

The next heuristic (H4) requires that a genotype is obtained from a Pareto optimal seed lot in terms of the corresponding probability and linkage phase ambiguity.

Heuristic H4 (Pareto optimal seed lot). Each genotype G is required to be obtained from a Pareto optimal seed lot $\mathcal{S}$ in terms

of probability and linkage phase ambiguity, among all seed lots available up to the respective generation.

Once a seed lot has been obtained from a crossing, it remains available in all subsequent generations. A targeted genotype may thus be contained in multiple seed lots. In such case it seems logical to use seeds from the seed lot which contains the desired genotype with the highest frequency, or which yields the lowest ambiguity. Although at first sight, this might appear to be an exact pruning criterion, forcing genotypes to be grown from specific seed lots may increase the cost required to provide a sufficient amount of seeds. Especially in case of a tight constraint on both the maximum number of crossings per plant and the number of seeds produced per crossing, heuristic H4 may thus in theory lead to suboptimal solutions, although this is expected to rarely happen in practice.

6.4.4 *Heuristic seed lot construction*

The number of possible offspring from a crossing grows exponentially with the number of (heterozygous) loci in the parents. Therefore, it can take a significant amount of time and memory to construct the entire seed lot. Although Gene Stacker includes several seed lot filters, this filtering step may also be time consuming. Therefore, we provide heuristics that reduce the number of haplotypes produced from the chromosomes of the crossed genotypes, by only considering promising crossovers. These heuristics (H5 and H5c) assume that a crossover is difficult to obtain and should therefore result in an obvious improvement.

Heuristic H5 (heuristic seed lot construction). Take a chromosome C with k loci of which $l \leqslant k$ are heterozygous with ordered indices $s = (\nu_1, \ldots, \nu_l)$. Also, take a haplotype H that is produced from C through $m < l$ crossovers between consecutive heterozygous loci $(\nu_{i_1-1}, \nu_{i_1}), \ldots, (\nu_{i_m-1}, \nu_{i_m})$. Split H into a series of $m+1$ corresponding stretches

$$\mathcal{H} = (S^H_{0,(\nu_{i_1})-1}, S^H_{\nu_{i_1},(\nu_{i_2})-1}, \ldots, S^H_{\nu_{i_m},k-1})$$

where each stretch $S^H_{i,j} \in \mathcal{H}$ originates from one of both haplotypes of C. For every stretch $S^H_{i,j}$ originating from the top haplotype C_1, i. e. $S^H_{i,j} = S^{C_1}_{i,j}$, the bottom haplotype C_2 contains an alternative stretch $S^{C_2}_{i,j} \neq S^H_{i,j}$ and vice versa. Produce only those haplotypes from C for which every stretch in $\mathcal{H}$ contains at least one desired allele which is not present in the alternative stretch.

Example 6.4. Consider the following chromosome

$$C = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

with respective ideotype chromosome

$$I = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Because all three loci in C are heterozygous, a total of eight haplotypes can be produced from this chromosome. However, when applying heuristic H5 half of the options are discarded. For example, the possible haplotype $H = [0\ 1\ 0]$ that is produced through a single crossover between the first and second locus, consists of two stretches

$$\mathcal{H} = (S^H_{0,0} = [0], S^H_{1,2} = [1\ 0])$$

where $S^H_{0,0} = S^{C_1}_{0,0}$ originates from the top haplotype C_1 of C, while $S^H_{1,2} = S^{C_2}_{1,2}$ originates from the bottom haplotype C_2. The alternative for the first stretch would be $S^{C_2}_{0,0} = [1]$ as found in the bottom haplotype C_2. Since $S^{C_1}_{0,0} = [0]$ does not contain any desired allele that is not present in the alternative stretch $S^{C_2}_{0,0} = [1]$ the crossover needed to produce H seems to be a waste of resources. The alternative haplotype $H' = [1\ 1\ 0]$ is likely a better target, because it more closely matches the ideotype, while also being more frequently observed in the offspring since no crossover is needed. Therefore, haplotype H is not produced when using heuristic H5.

Similarly, the possible haplotype $H = [0\ 1\ 1]$ contains three stretches

$$\mathcal{H} = (S^H_{0,0} = S^{C_1}_{0,0} = [0], S^H_{1,1} = S^{C_2}_{1,1} = [1], S^H_{2,2} = S^{C_1}_{2,2} = [1])$$

originating from alternating haplotypes of C and obtained through two crossovers—one between each consecutive pair of loci. The second and third stretch contain a desired 1-allele not present in the alternative stretch found at the other haplotype of C, but again this is not the case for the first stretch, meaning that aiming for the first crossover is likely an unnecessary waste of resources. The alternative haplotype $H' = [1\ 1\ 1]$ seems to be a better choice, since it requires only a single crossover and fully matches the target haplotype of the ideotype chromosome.

Due to the same reasoning haplotypes $[0\ 0\ 0]$ and $[1\ 0\ 0]$ are also not considered here when heuristic H5 is enabled. △

Heuristic H5c (consistent heuristic seed lot construction). This heuristic is a stronger version of H5 that requires consistent improvement within all stretches towards a fixed haplotype of the corresponding ideotype chromosome.

Example 6.5. Consider the same chromosome

$$C = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

as in the previous example but now with a respective heterozygous ideotype chromosome

$$I = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

As before, the haplotype $H = [0\ 1\ 0]$ can be produced from C through a crossover between the first and second loci. Even when heuristic H5 is enabled, this haplotype would be considered here, because the first stretch $S^{H}_{0,0} = S^{C_1}_{0,0} = [0]$ contains a desired 0-allele not found in the alternative stretch $S^{C_2}_{0,0} = [1]$, and the second stretch $S^{H}_{1,2} = S^{C_2}_{1,2} = [1\ 0]$ contains a 1-allele that is desired at the second locus of the ideotype chromosome but not found in $S^{C_1}_{1,2} = [0\ 1]$. Thus, both stretches contain a desired allele not present in the alternative stretch and therefore heuristic H5 will consider this haplotype. However, if we look closer, we notice that $S^{C_1}_{0,0} = [0]$ is better than the alternative $S^{C_2}_{0,0} = [1]$ in terms of the top haplotype $I_1 = [0\ 0\ 1]$ of I, while the second stretch $S^{C_2}_{1,2} = [1\ 0]$ is only advantageous over the alternative stretch $S^{C_1}_{1,2} = [0\ 1]$ when aiming for the bottom target haplotype $I_1 = [1\ 1\ 1]$. Such inconsistencies are not allowed by heuristic H5c, which will therefore discard the haplotype $H = [0\ 1\ 0]$. This makes sense because the best way to obtain the top target haplotype $I_1 = [0\ 0\ 1]$ in the offspring is simply to aim for no crossovers at all, since the top haplotype $C_1 = [0\ 0\ 1]$ of C is already equal to this target. Likewise, the haplotype $C_2 = [1\ 1\ 0]$ better matches the bottom target haplotype $I_2 = [1\ 1\ 1]$ as compared to $H = [0\ 1\ 0]$, and will be more frequently observed without requiring any crossovers.

For the given chromosome and heterozygous ideotype, heuristic H5 only discards haplotypes [0 0 0] and [1 0 0], while H5c will also not produce [0 1 0] and [0 1 1]. △

For a homozygous ideotype, H5c degenerates to H5. To be able to compute linkage phase ambiguities, a heuristically constructed seed lot $\mathcal{S}$ is further extended to include all phase-known genotypes with the same allele frequencies as any genotype already contained in $\mathcal{S}$. Heuristics H5 and H5c also provide an option to limit the number of simultaneous crossovers per chromosome, to further reduce the number of generated haplotypes if necessary.

6.4.5 *Approximate population size bound*

Finally, heuristic H6 computes an approximate lower bound on the population size of any completed extension of a given partial scheme, based on the probabilities of those crossovers that are necessarily still required to obtain the ideotype.

Heuristic H6 (approximate population size bound). For each chromosome I of the ideotype $\mathcal{I}$, having n_I loci, the set of desired stretches of length two is defined as

$$\mathcal{D}_I = \left\{ S^H_{i,i+1}; H = I_1 \vee I_2 \;\&\; 0 \leqslant i < n_I - 1 \right\}.$$

From $\mathcal{D}_I$ all stretches that occur in the respective chromosome of a parental genotype $G \in \mathcal{G}$ are discarded. For each retained stretch $S^H_{i,i+1}$ a crossover is necessarily required between loci i and $i+1$ to obtain the ideotype. Now, given a partial scheme, it is checked, for all chromosomes, which of the crucial stretches are not yet present in any genotype occurring in this scheme. The sum of the minimum population sizes required to obtain each of the corresponding crossovers is used as a lower bound for the increase in total population size of any completed extension of this scheme.

Example 6.6. Consider a problem with two initial parents

$$G^1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \quad \text{and} \quad G^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

and ideotype

$$\mathcal{I} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

The set of desired stretches of length two for the single involved chromosome is then defined as

$$\mathcal{D} = \left\{ S_{0,1} = [1\ 1], S_{1,2} = [1\ 1] \right\}.$$

The first of these two desired stretches is already present in initial parent G^1, but the second is not yet found in any of the two parents. Therefore, a crossover between the second and third loci is necessarily required somewhere in the scheme, and as long as it has not occurred the minimum additional population size that will be required to aim for this crossover is used to compute a lower bound on the eventual total population size.

Suppose that the second and third loci are tightly linked at a distance of 3 cM, meaning that there is about a 3% chance of observing a crossover between these loci. If, for example, we want to be 95%

Preset	Enabled heuristics	Dual run
Best	none	
Better	H0, H1a, H2a, H3s1	✓
Default	H0, H1a, H2a, H3s1, H4, H5, H6	✓
Faster	H0, H1b, H2b, H3s2, H4, H5c, H6	✓
Fastest	H0, H1b, H2b, H3, H4, H5c, H6	

Table 6.1: Heuristic presets combining well-chosen heuristics.

sure to find the corresponding crucial stretch among the offspring of some crossing, this will require a population size of at least

$$\left\lceil \frac{\log(1-0.95)}{\log(1-0.015)} \right\rceil = 199$$

for that crossing. In other words we know that completing any scheme that still has avoided this expensive crucial crossover will require to grow and screen at least about 200 more plants, which may allow earlier pruning of branches in the search tree once some solutions have already been found. △

It might seem that heuristic H6 implements an exact bound but this is not guaranteed as Gene Stacker computes a joint population size when targeting multiple genotypes among offspring obtained from the same seed lot (see appendix B.2). It is therefore possible that multiple crucial stretches are simultaneously obtained with a lower total cost. However, it is expected that this will rarely occur, which makes heuristic H6 a nearly exact bound.

6.4.6 *Presets*

Several well-chosen combinations of heuristics provide tradeoffs between solution quality and execution time. Presets are named *best*, *better*, *default*, *faster* and *fastest*, ordered by the amount and restrictiveness of the applied heuristics (table 6.1). In the *default* setting some less restrictive heuristics are applied compared to those enabled when switching to presets *faster* and *fastest*. On the other hand, preset *better* drops some heuristics and preset *best* does not apply any (optional) heuristics at all. Presets *better*, *default* and *faster* perform two runs as they apply one of the dual run heuristics H3s1 or H3s2, while preset *fastest* applies H3 in a single run.

6.5 RESULTS AND DISCUSSION

This section presents results of applying Gene Stacker to both generated and real stacking problems. First, we highlight some advantages of the extended DAG model. Then, the power of the applied optimization strategy in combination with the proposed heuristics is assessed. We conclude by providing some practical guidelines for users of Gene Stacker. Results are compared to those obtained by Canzar and El-Kebir (2011). Their method, further referred to as CANZAR, minimizes the total population size, number of generations, and total number of crossings. As minimizing the number of crossings is not explicitly considered as an objective in Gene Stacker, only the schemes with the lowest total population size among those with the same number of generations, produced by CANZAR, were selected for comparison with Gene Stacker.

Experiments with CANZAR were run on the SURFsara Lisa computing system (see `https://goo.gl/OGzV4c`) by Mohammed El-Kebir.

6.5.1 *Advantages of the extended model*

We first highlight some advantages of our extended DAG model, based on two constructed examples and a complex real stacking problem from cotton.

Constructed examples

Consider an example with two heterozygous parental genotypes G^1, G^2 and a heterozygous ideotype $\mathcal{I}$:

$$G^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$
$$G^2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},$$
$$\mathcal{I} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

The distance between the loci on the second chromosome is 31 and 42 cM, respectively. Five solutions were reported when running Gene Stacker in *default* mode, setting an overall success rate of $\gamma = 0.95$, and allowing a maximum of 4 generations and 10% overall linkage phase ambiguity (appendix B.3; figures B.1 to B.5).

Figure 6.8 (left) shows the best non-ambiguous three generation long scheme obtained by Gene Stacker, with a total population size

of 275, as well as the respective best three generation long solution found by CANZAR (right), which has a higher total population size of 363. The leftmost target aimed for in the penultimate generation of the latter scheme has a linkage phase ambiguity of 23.1%, which is not reported nor taken into account during optimization by CANZAR, while Gene Stacker's solution is guaranteed to be non-ambiguous. Gene Stacker provides a way to avoid such high ambiguities by carefully monitoring them and considering ambiguity as an additional objective to be minimized.

This example also shows how computing joint population sizes when simultaneously targeting multiple genotypes among the offspring grown from the same seed lot may significantly reduce the total population size (seed lot S3 in figure 6.8). This approach enabled Gene Stacker to find an alternative scheme with a reduction of more than 24% in the total population size as compared to the scheme constructed by CANZAR.

Another advantage of representing plants and seed lots with distinct nodes is that (re)use of plants and seeds is differentiated. Gene Stacker only allows crossings with plants from the same generation, which is justified by the fact that almost all field crops flower only once, for a short time. Also for crops that flower multiple times or for a longer period—such as tomato—crossings with plants from distinct generations are usually not considered because of the high logistic impact. To repeatedly cross over multiple generations, it is thus preferred to reproduce the respective genotype, for example by regrowing it from remaining seeds. In such case, the corresponding cost is accounted for by Gene Stacker. Note that this does not limit the flexibility of Gene Stacker's model but ensures that the computed cost of the constructed schemes closely reflects plant breeding practice.

The second example has specifically been constructed to show the advantage of modelling multiple chromosomes. It consists of the following two parental genotypes and ideotype:

$$
\begin{aligned}
G^1 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \\
G^2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\
\mathcal{I} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}.
\end{aligned}
$$

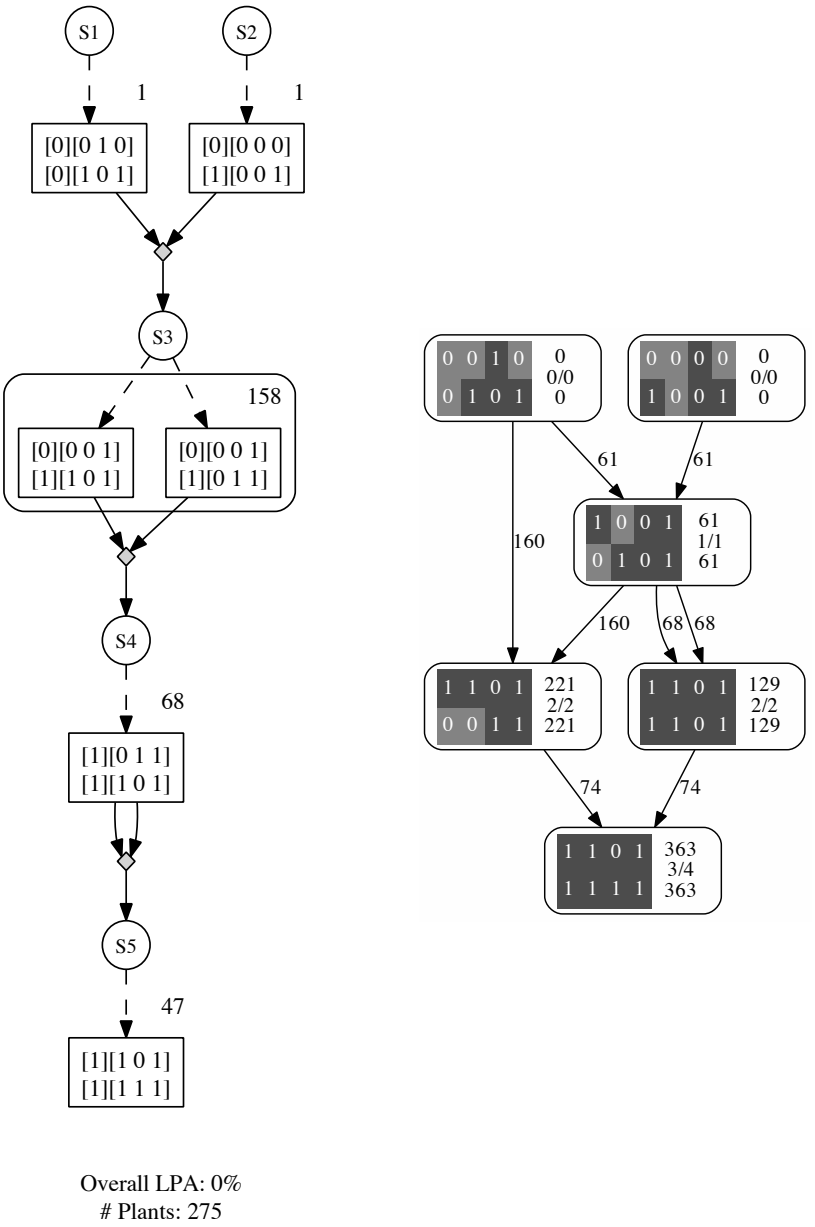


Figure 6.8: Best non-ambiguous three generation long scheme obtained with Gene Stacker (left) in *default* mode for the first constructed example, as compared to the respective best three generation long solution reported by CANZAR (right).

We do not need to specify distances between consecutive loci on the same chromosome, as each chromosome contains only one locus of interest. Running Gene Stacker with any preset and $\gamma = 0.95$ resulted in the scheme from figure 6.9 (left) which consist of a single generation in which a single crossing is performed. It is possible to immediately obtain the ideotype from this crossing because the order of haplotypes within a chromosome is arbitrary, which is taken into account when computing the probability of observing a genotype among the offspring (see appendix B.1).

Previous methods, including CANZAR, modelled only a single chromosome and specified a recombination rate of 0.5 between loci that actually reside on different chromosomes. This requires to fix an arbitrary order of haplotypes in each actual chromosome and artificially increases the complexity of the problem. Figure 6.9 (right) shows Gene Stacker's solution for the same example when combining all loci on such artificial chromosome. This scheme is significantly worse: it has an additional generation and a much higher total population size. Although this example was specifically constructed and is somewhat extreme in the sense that it has six loci on six different chromosomes, it clearly shows the general benefits of explicitly modelling multiple chromosomes.

Dealing with tight constraints

Tight constraints might apply for specific crops. For example, cotton plants can be used for two crossings only (or one selfing) and each crossing yields a small amount of about 250 seeds. With the extended model such important operational constraints can easily be taken into account. Crossings are performed multiple times if necessary to provide a sufficient amount of seeds, where sometimes several duplicates of the same genotype are needed to be able to make all crossings. Population sizes are computed in such way that at least the required number of occurrences of each targeted genotype is expected among the offspring (see appendix B.2).

We now consider a real example from cotton with six parental genotypes, 11 loci spread across five chromosomes and a heterozygous ideotype (for a full description, see appendix B.4). The overall success rate was set to $\gamma = 0.95$, and the number of generations, number of plants per generation, and overall linkage phase ambiguity were limited to 5, 5000, and 10%, respectively. We applied a time limit of 24 hours. The number of crossings per plant and seeds obtained per crossing were set to 2 and 250, respectively, to precisely reflect the tight constraints of cotton breeding.

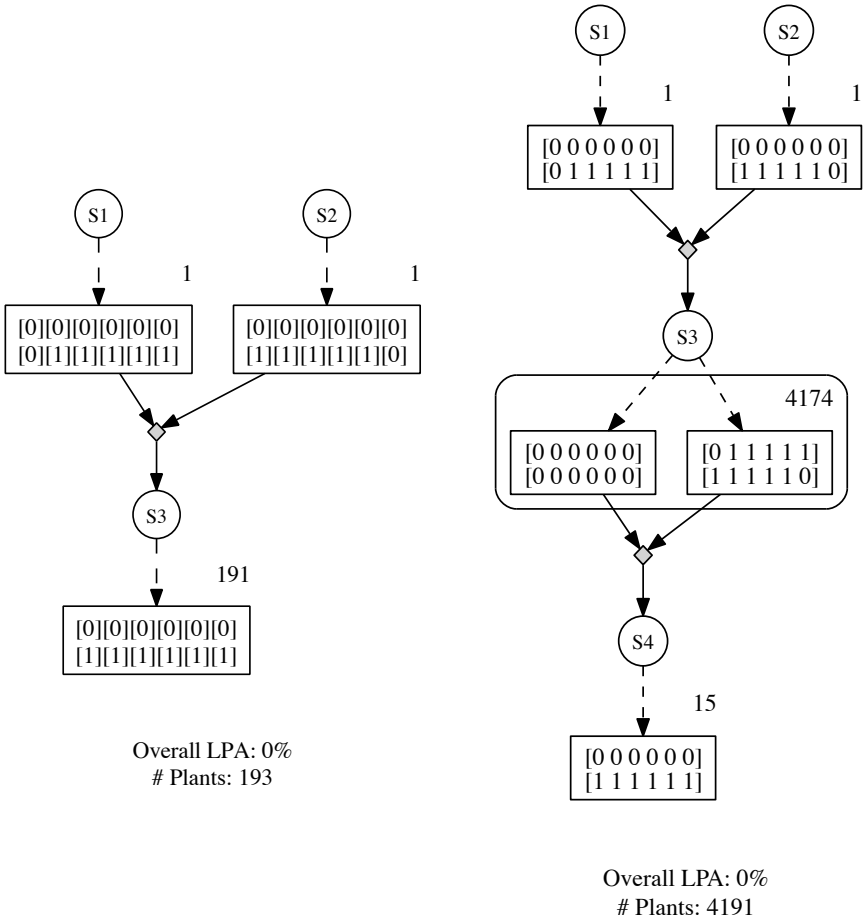


Figure 6.9: Best scheme obtained for the second constructed example when explicitly modelling multiple chromosomes (left), as compared to the best solution found when combining all loci on one artificial chromosome (right). In the latter case a crossover rate of 0.5 is specified between pairs of consecutive loci that actually reside on different chromosomes (in this example between all loci).

Running Gene Stacker with preset *fastest* completed after 2 hours and 15 minutes, and reported four solutions with 3–5 generations, a total population size of 7256–1077 and an overall linkage phase ambiguity of 0–3.14% (see appendix B.5; figures B.6 to B.9). All other presets ran out of memory (64 GB). When restricting the number of generations to four instead of five, preset *faster* reported a different four generation long solution, that has a lower total population size (1400) than the respective scheme found by preset *fastest* (1534) before being interrupted when the time limit of 24 hours had been exceeded (appendix B.5; figure B.10). All solutions contain at least one crossing that is performed multiple times, to produce enough seeds, and/or a genotype of which multiple duplicates are grown, to be able to complete all crossings. It was not possible to obtain solutions within the constraints using CANZAR as this method does not provide a way to accurately impose and work around such operational constraints.

6.5.2 *Optimization power and heuristics*

We first explore the limits of the optimization strategy and the power gained by applying additional heuristics, based on experiments with a large number of randomly generated problem instances. Then, the obtained quality-runtime tradeoff is assessed for various complex, real stacking problems.

Limits of the optimization strategy

We experimented with a variety of 240 randomly generated stacking problems. Of these, 120 have a homozygous ideotype and the remaining 120 have a heterozygous ideotype. All instances have 4–14 loci, taking steps of two, and 20 instances were created for every number of loci and for both types of ideotype. Each instance has been independently generated by (i) picking a random number of 1–8 chromosomes, limited by the number of loci; (ii) randomly assigning each locus to one of the available chromosomes, with a minimum of 1 locus per chromosome; (iii) setting a random distance of 1–50 cM between pairs of consecutive loci on the same chromosome; (iv) randomly creating 2–8 parental genotypes, where each allele is set to 1 or 0 with equal probability; and (v) generating a random ideotype. The haplotypes of the ideotype's chromosomes were created by copying alleles from one of both haplotypes of the respective chromosome of a randomly chosen parental genotype (independently for every locus). To obtain a homozygous ideotype, one haplotype is created for each chromosome and included twice.

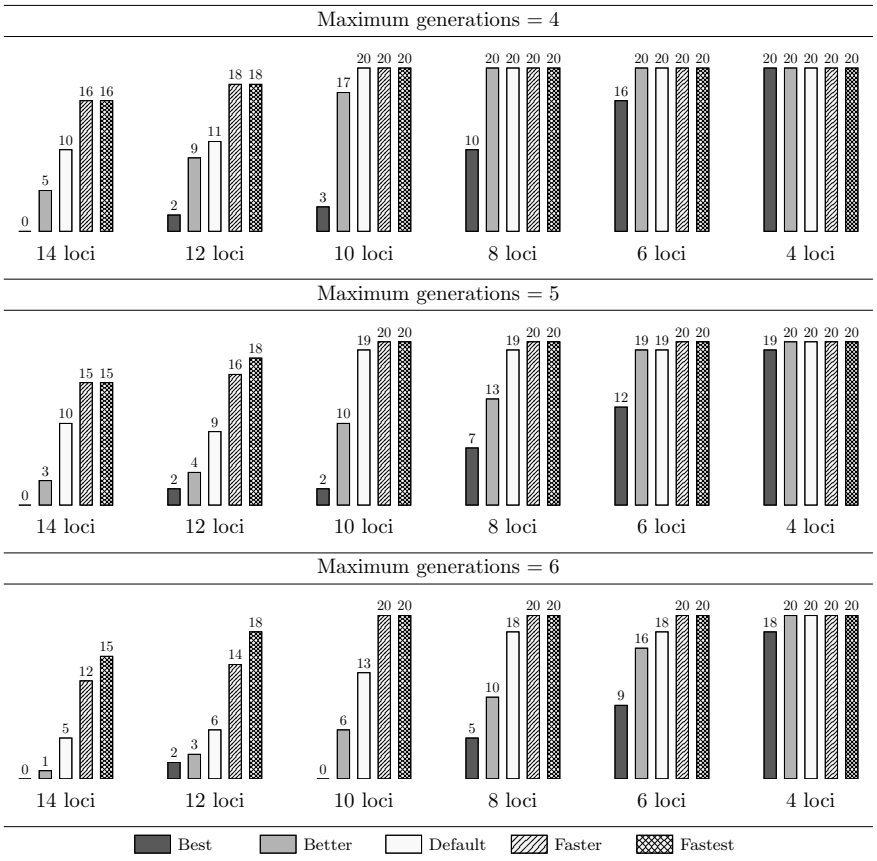


Figure 6.10: This figure indicates the number of randomly generated instances with a *homozygous* ideotype for which the different presets of Gene Stacker completed within the applied time limit of 24 hours. Experiments were repeated with a maximum of 4–6 generations. Instances have 4–14 loci, spread across 1–8 chromosomes, and 2–8 parental genotypes. In total, 20 instances were generated for each number of loci.

For heterozygous ideotypes, two independent haplotypes are created and combined for every chromosome.

Figure 6.10 shows results of running each preset of Gene Stacker on the 120 instances with a homozygous ideotype. We repeated all experiments with a maximum of 4, 5 and 6 generations, and applied a runtime limit of 24 hours, together with an overall success rate of $\gamma = 0.95$ and a maximum of ten thousand plants per generation, four crossings per plant, five thousand seeds per crossing, and 20% overall linkage phase ambiguity. For every combination of the maximum number of generations (rows), the number of loci (columns) and the applied preset (bars) it is reported for how many out of 20 instances Gene Stacker completed within the time limit of 24 hours.

Without applying any heuristics (preset *best*), Gene Stacker solves only 42.5%, 35% and 28.34% of all instances when limiting the number of generations to 4, 5 and 6, respectively. Interestingly, solutions are obtained for about 95% of all instances when applying all heuristics (preset *fastest*) regardless of the limit on the number of generations. As expected—and desired—the power of the other presets (*better*, *default*, *faster*) lies somewhere in between. The problem complexity obviously increases with the number of loci as well as the maximum number of generations. Without any heuristics, Gene Stacker solved almost no problems with more than 8 loci: solutions were obtained for less than half of the instances when the number of loci exceeded 8, 6 and 4 with a limit of 4, 5 and 6 generations, respectively. Yet, Gene Stacker can cope with many more complex problems with up to at least 14 loci using the proposed heuristics. Of course, using these heuristics may yield worse Pareto front approximations, so it is preferred to enable them only if necessary to find solutions within reasonable time. In this way, the heuristics offer a convenient quality-runtime tradeoff and allow to obtain (approximate) solutions for complex problems.

Figure 6.11 shows similar results for the 120 instances with a heterozygous ideotype. It is clear that these are generally more complex, as compared to those with a homozygous ideotype, since significantly fewer instances were solved within the time limit. One reason for this higher complexity is that each heterozygous chromosome in the ideotype contains two different target haplotypes, i. e. two competing goals, that have to be obtained simultaneously. Also, the heuristics are less effective for heterozygous ideotypes. For example, improvement towards any of both haplotypes of a heterozygous ideotype chromosome is rewarded. Therefore, heuristics based on such improvement are less effective in case of two distinct target haplotypes in a single chromosome.

Without applying any heuristics, Gene Stacker now solves 22.5–37.5% of all instances for a varying limit on the number of generations. Less than half of the instances were solved when the number of loci exceeded 4–6. When all heuristics are enabled, solutions are obtained for 65–72.5% of the instances (for less than half of the instances when exceeding 10 loci). Although the currently proposed heuristics are clearly less powerful when aiming for a heterozygous ideotype, they allowed to find solutions for many complex problems with up to 10 loci. Nevertheless, the challenge remains to develop better heuristics in this respect.

We conclude that the applied optimization strategy can effectively be used to find solutions for a wide range of stacking problems. Without extra heuristics, some smaller problems with 4–8 or 4–6

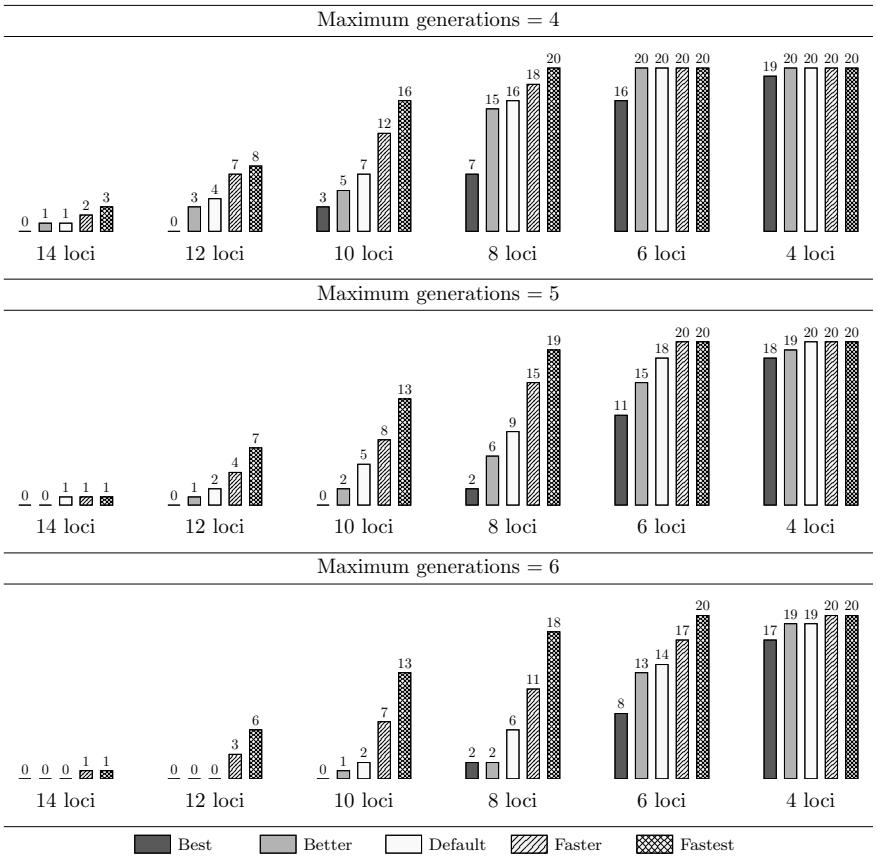


Figure 6.11: This figure indicates the number of randomly generated instances with a *heterozygous* ideotype for which the different presets of Gene Stacker completed within the applied time limit of 24 hours. Experiments were repeated with a maximum of 4–6 generations. Instances have 4–14 loci, spread across 1–8 chromosomes, and 2–8 parental genotypes. In total, 20 instances were generated for each number of loci.

loci in case of a homozygous or heterozygous ideotype, respectively, can already be tackled, depending on the maximum number of generations. To deal with more complex problems, additional heuristics are required. The proposed heuristics allow to obtain (approximate) solutions for problems with up to at least 10–14 loci.

Quality-runtime tradeoff

Now we assess the quality-runtime tradeoff obtained by applying different combinations of heuristics for real stacking problems, originating from tomato and rice breeding (for a full specification, see appendix B.4). For all experiments, we set an overall success rate of $\gamma = 0.95$, and restricted the number of generations and plants per generation to 5 and 5000, respectively. The amount of seeds pro-

duced per crossing and maximum number of crossings per plant were set to reflect the specific properties of each crop, as specified below. We selected only solutions with zero linkage phase ambiguity, and report approximated Pareto fronts in terms of the total population size and number of generations.

The two considered stacking problems from tomato both consist of the same four parental genotypes with eight loci spread across six chromosomes. The first (Tomato-1) and second (Tomato-2) example have a homozygous and heterozygous ideotype, respectively. Tomatoes can easily be crossed several dozens of times and every crossing yields a large number of seeds. Therefore, we set the maximum number of crossings per plant and the amount of seeds obtained from one crossing to 24 and 20.000, respectively. A time limit of 12 hours was imposed, after which the algorithms were interrupted and the solutions found until then were inspected.

Figure 6.12 (top left) shows the Pareto front approximations obtained for Tomato-1 with Gene Stacker, using presets *default*, *faster*, and *fastest*, as well as CANZAR. Gene Stacker and CANZAR obtained exactly the same scheme with four generations. The small difference in the reported population size is explained by the fact that both methods follow a different approach to derive a success rate per targeted genotype (γ') from the desired overall success rate (γ). Solutions with five generations were also found. Those reported by Gene Stacker have a lower population size as compared to the one obtained by CANZAR, even when applying preset *fastest* which completes after only 28 seconds. Presets *default* and *faster* reported exactly the same solutions, and the five generation long scheme found here improves over the respective scheme obtained by preset *fastest*. Yet, these two presets took significantly more time (about 6–8 hours). These results again show how the proposed heuristics provide convenient tradeoffs between solution quality and execution time, and that they are capable of finding good solutions for a complex, realistic problem within reasonable time. CANZAR was interrupted when exceeding the time limit of 12 hours.

Similar results for Tomato-2 are presented in figure 6.12 (top right) where only preset *fastest* has been applied since the other presets ran out of memory (64 GB). Gene Stacker completed in about five hours while CANZAR was interrupted when the time limit had expired. Three solutions were reported by Gene Stacker with 3–5 generations and CANZAR obtained two solutions with 4–5 generations. The four generation long schemes reported by both methods slightly differ (results not shown) but have approximately the same total population size. Conversely, Gene Stacker found a somewhat better scheme with five generations and an additional solution with

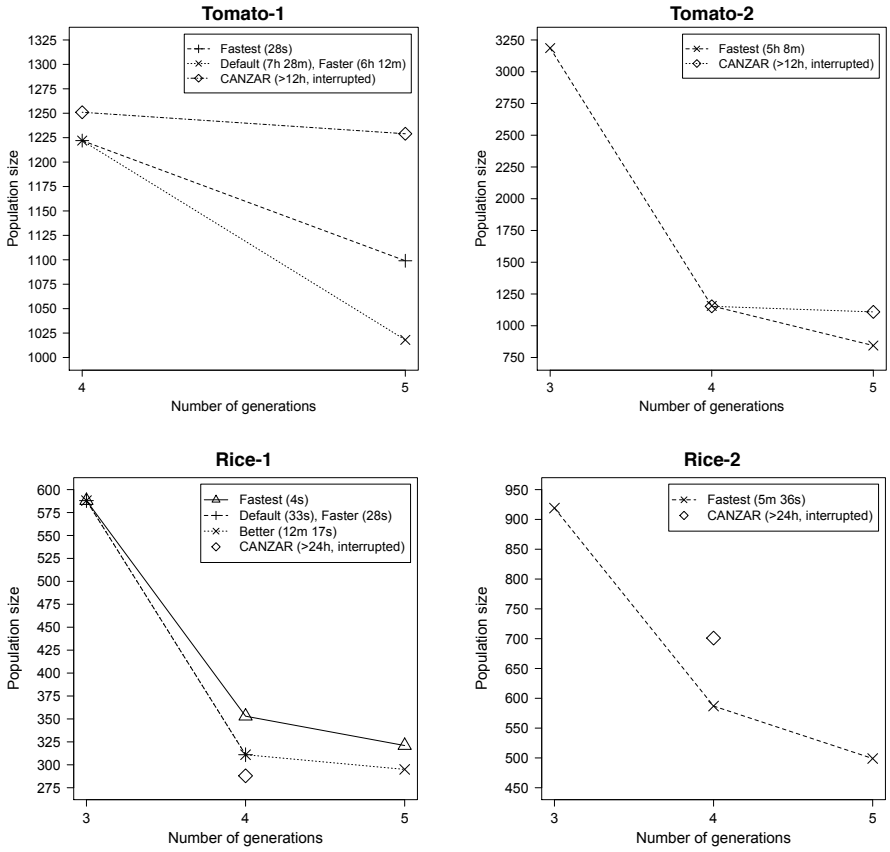


Figure 6.12: Pareto front approximations of real stacking problems from tomato and rice: (top left) first example from tomato (Tomato-1; homozygous ideotype); (top right) second example from tomato (Tomato-2; heterozygous ideotype); (bottom left) first example from rice (Rice-1; homozygous ideotype); (bottom right) second example from rice (Rice-2; heterozygous ideotype). Full descriptions of the example problems are provided in appendix B.4.

only three generations. The difference in runtime, as compared to Tomato-1, and the fact that all other presets ran out of memory, again confirm that with the current heuristics it is more difficult to solve stacking problems with a heterozygous ideotype. Yet, the heuristics made it possible to find three good solutions within a few hours, using a transparent optimization strategy.

We also experimented with two other examples, originating from rice breeding. Both consist of the same eight parental genotypes with ten loci spread across six chromosomes. Again, the first (Rice-1) and second (Rice-2) example have a homozygous and heterozygous ideotype, respectively. About 300 seeds are obtained from each crossing and rice plants can be crossed no more than 5 times. For these examples, a time limit of 24 hours was set.

Figure 6.12 (bottom left) shows results for Rice-1 obtained with Gene Stacker, using presets *better*, *default*, *faster*, and *fastest*, as well as CANZAR. Preset *fastest* completed after only four seconds and reported three solutions with 3–5 generations. Presets *default* and *faster* terminated after about 30 seconds and found a better scheme that dominates both the four and five generation long schemes obtained by preset *fastest*. Preset *better* completed after about 12 minutes and found an additional five generation long scheme with a slightly lower total population size. These results again show how the heuristics offer a convenient quality-runtime tradeoff. CANZAR did not complete within the time limit of 24 hours but was able to obtain a single scheme with four generations that dominates all four and five generation long schemes obtained by Gene Stacker. It is inevitable that the heuristics sometimes make wrong decisions in which case valuable parts of the search space may not have been explored. In this specific example, heuristic H0 (included in all presets except *best*) removed a parental genotype that is needed to find the slightly better scheme obtained by CANZAR. Still, results are close to those of CANZAR, especially when applying presets *faster*, *default* or *better*, a significant speedup is obtained, and an additional solution with only three generations is found.

Similar results for Rice-2 are shown in figure 6.12 (bottom right) where only preset *fastest* has been applied as the other presets either ran out of memory or did not find any solutions within the time limit. Gene Stacker completed after 5–6 minutes while CANZAR was interrupted after exceeding the time limit of 24 hours. Three solutions were reported by Gene Stacker, with 3–5 generations. CANZAR found a single solution with four generations and a higher population size than the respective scheme obtained by Gene Stacker. Again, the runtime and memory footprint of Gene Stacker is significantly higher for this problem with a heterozygous ideotype as compared to Rice-1 which has a homozygous ideotype. Yet, preset *fastest* outperforms CANZAR and is able to provide a valuable approximation of the Pareto front within a few minutes.

6.5.3 *Practical guidelines*

Based on our findings we propose the following practical guidelines for using Gene Stacker. It is advised to first try the default settings, specifying the required parameters (maximum number of generations and overall success rate) and those constraints that are important for the specific application, such as the number of seeds produced from a crossing and maximum number of crossings per plant, with a reasonably high runtime limit (e. g. 24 hours). If Gene

Stacker is too slow or requires too much memory, consider setting additional or tighter constraints (e. g. maximum plants per generation, maximum overall linkage phase ambiguity, ...) and/or using preset *faster* or *fastest*. The latter may yield worse solutions which should be avoided when possible. In case the default setting is more than fast enough consider running presets *better* and *best* as well to check whether this produces better schemes, as the heuristics might have missed something. Usually, differences between the latter presets and the default setting are very small (if any) except for the runtime which is significantly increased. More details and practical examples are given at `http://genestacker.ugent.be`.

In case QTL intervals need to be stacked one can use flanking markers to delimit the target locus. The Tomato-1 problem (appendix B.4) is a case in point. On the sixth chromosome, a small region of 10 cM has been identified in which a target gene is located. In this setting it is necessary to make sure that the required haplotype is present in at least one of the parents, and to verify that it is maintained throughout the crossing scheme. There always remains a small risk of a double cross-over within the interval in a single generation which one can either ignore or monitor by saturating the interval with additional markers.

6.6 CONCLUSIONS

The proposed transparent, flexible, and easily extensible approach to marker-assisted gene pyramiding was confirmed to be feasible in combination with heuristics to address realistic, complex stacking problems with up to at least 10–14 loci, while taking into account important operational breeding constraints. Carefully designed heuristics allow to find better or additional solutions within reasonable time as compared to previous methods. The proposed heuristics are certainly not perfect nor complete. For example, they are less effective for problems with a heterozygous ideotype. Still, even for these more complex problems Gene Stacker is able to find approximate solutions with high practical value within reasonable time. Future work may include the design of additional or improved heuristics as well as extension of the ideas applied in Gene Stacker for a more general plant breeding context that also addresses complex traits and conservation of genetic background.

ACKNOWLEDGEMENTS

We thank Mohammed El-Kebir and Stefan Canzar for their kind cooperation by running their algorithm on our test cases. This allowed for an interesting comparison between our methods and made a major contribution to the discussion.

IMPLEMENTATION AND HARDWARE

Gene Stacker is implemented in Java 7 and experiments have been performed on the UGent HPC infrastructure, using computing nodes with a 2.4 GHz quad-socket octa-core AMD Magny-Cours processor having a total of 32 cores and 64 GB RAM. Gene Stacker is freely available at http://genestacker.ugent.be. Version 1.6 was used for all experiments. The website also contains user documentation and examples.

Part IV

CONCLUSIONS

7

CONCLUSIONS AND FUTURE PERSPECTIVES

Plant breeding is as old as agriculture itself. For thousands of years, farmers selected seeds from good looking plants and stored them for planting in the next season. As such—mostly unknowingly—these early breeders created many landraces that are highly adapted to their local environment. After the Middle Ages scientist started to unravel the sexuality of plants and were able to perform artificial crossings providing an additional source of variability for selection. Not much later, commercial plant breeding companies were established that continuously improved plant varieties through repeated crossing and selection based on observable characteristics (the phenotype) and sold their enhanced seeds to the farmers.

Starting with the work of the famous Gregor Mendel in the 19th century, several important discoveries followed that revealed the underlying genetics responsible for the diversity of characteristics observed in all living organisms, including plants. Methods were developed to extract DNA fragments to get a view on the genetic architecture—the genotype—of an animal or plant. This genetic information can be used by breeders to make better decisions, as their ultimate goal is to gather a maximum of beneficial genes in a single plant variety so that it will maximally develop desirable traits. Especially during the last few decades genotyping costs significantly decreased to a point where they are no longer limiting, and now the main question is how to optimally use genetic data in practical breeding schemes.

In this thesis we applied discrete optimization algorithms to solve several problems related to genomics-assisted breeding. In the first two chapters we provided a broad background of both of these disciplines, describing important concepts needed to understand the breeding problems addressed and optimization techniques applied in the following chapters.

Next, in chapter 3 we presented the JAMES framework, an object-oriented Java framework for discrete optimization with local search metaheuristics, that is used in subsequent chapters to solve multiple problems with the same optimization engines. JAMES differentiates from existing Java metaheuristics frameworks in its focus on local searches, which had significant impact on its design and core features, such as an efficient movement-based evaluation mechanism.

A computational comparison with other frameworks showed that our efforts clearly paid off, as the results were very much in favour of JAMES in terms of both execution time and memory usage.

One very appealing direction for further development of JAMES would be to also include population-based algorithms. Although this may seem a bit contradictory to the fact that JAMES has been built specifically for efficient application of local searches, the design of the framework does not yield any limitations for also including population-based methods. The latter do not need sophisticated low-level features such as movement-based evaluation, but these can easily be ignored, and any new high-level components needed for population-based algorithms—such as interfaces defining crossover, mutation, and selection operators for a genetic algorithm—can easily be added on top of the core design. Not only would it be very valuable to have more types of optimization algorithms in the same framework, so that users can select the most effective method for their application, in addition, this for example also allows to compose advanced hybrid methods, such as a genetic algorithm that applies a local search as its mutation operator. In the latter case, the specific features of JAMES for use in local searches can again be exploited. One may wonder if it would not also be easy to add efficient local searches to other, existing frameworks currently focused at population-based methods. We are convinced that this would be a much more difficult task, requiring a significant refactoring of the core design of these frameworks, because they lack important low-level features that are not easily introduced, though essential for efficient local searches.

Since much of the biodiversity that exists in cultivated plant species is not directly used for agricultural purposes, breeders often rely on the availability of large genetic resources stored in gene banks to keep improving their products. These collections contain a huge amount of varieties—including historical landraces, modern cultivars, and wild relatives—of all major crops, and are very useful resources for breeders and plant researchers in general. Due to their size, however, the entire collections cannot be characterized in full detail, nor effectively utilized or distributed. Therefore, smaller core collections are often composed that represent the diversity of the full collection with minimum redundancy, and allow efficient access to large genetic resources for future crop improvement.

In chapter 4 we introduced Core Hunter 3: a flexible tool for multi-purpose core subset selection. Core Hunter samples diverse subsets whose entries have a highly dissimilar genetic or phenotypic profile, as well as cores that maximally represent each individual plant from the full collection. In addition, Core Hunter can maximize

allelic richness, for example to avoid loss of rare alleles (rare gene variants). To optimize the chosen evaluation measure, or a weighted index that balances multiple measures, Core Hunter uses fast local search algorithms from the JAMES framework. Core Hunter is extremely flexible, and new criteria can easily be introduced without the need to change the underlying optimization engine. Moreover, it outperforms other algorithms that were developed to construct specific types of core collections. Although core subset selection was introduced in the context of gene bank management, the potential of Core Hunter reaches far beyond this application. For example, it can also be used to compose diverse training material for a genomic prediction model (see below).

Currently, Core Hunter focuses on fixed-size core sampling, where the size is specified by the user. Although previous versions of Core Hunter allowed to specify a size range and favoured smaller cores, we believe that minimizing the core size might not always be desired and that the employed evaluation measures are not appropriate to compare cores of different sizes. Therefore, we removed this feature. A fairly common requirement however is to construct a core of minimum size that still covers all or most of the observed alleles or traits. Specific algorithms have been proposed for this purpose, but experiments with the first version of Core Hunter revealed the potential of local search algorithms to improve on these existing methods. To properly address this kind of variable-size core sampling within Core Hunter we could for example start a local search from the full collection, minimizing the core size while taking into account that the coverage cannot drop below a user specified threshold. In any case, we are convinced that fixed- and variable-size core sampling should be treated as separate problems, with different optimization objectives.

Another unexplored potential of Core Hunter is core sampling based on a combination of genetic data and phenotypic traits. Although it is currently already possible to load both genotypes and phenotypes, and to optimize a weighted index that balances genetic and phenotypic diversity, the main challenge here is to determine appropriate weights. The most straightforward approach would be to assign equal weights to both data sources, but they might overlap—perhaps even asymmetrically—in which case a bias could be introduced towards diversity at certain parts of the genome. It would be very interesting to experiment with Core Hunter to analyse datasets for which both genetic data and several phenotypes have been recorded, and to assess the need for intelligent weights that optimally exploit and balance both data sources.

Once the starting material has been composed, plant breeders typically go through several generations of crossing and selection to accumulate beneficial genes from different parents in a single enhanced variety. Unfortunately, quantitative traits—such as yield, height or size—usually have a complex genetic architecture where many genes spread across the genome have a small additive effect on the expressed value. Scientists are working hard to identify all responsible genes for important traits in major crops and vegetables. If all these causal genes were known, breeders could focus on concentrating them in a single genotype to improve the expressed phenotype. However, because so many genes are involved, it is difficult to find them all.

A practical solution is to characterize the genome with dense genetic markers from which an individual's breeding value can be predicted, based on a training population for which both genotypes and phenotypes are available, in the hope that the effect of most causal genes will be picked up by a marker in its vicinity. This technique, known as genomic selection, is one of the major marker-assisted selection trends in modern molecular breeding. It has the advantage of accelerating the selection cycle—the costly and time consuming phenotypic evaluation can be skipped or at least postponed—and improves selection accuracy, especially for traits that are difficult to observe. However, it is known that genomic selection also more rapidly depletes diversity due to which, although short-term gain can be significantly increased, long-term improvement is hindered.

Therefore, in chapter 5 we evaluated several existing and new strategies for long-term genomic selection, that balance gain and diversity. Existing strategies either aim to maximize gain under a predefined inbreeding rate (genomic optimal contributions selection; GOCS) or amplify the predicted effect of rare favourable alleles (weighted genomic selection; WGS) to avoid that they are lost due to selection. Our simulations indicate that both methods have inherent limitations: GOCS does not control inbreeding at the target level because it ignores an important component of the inbreeding rate that occurs only under selection, and WGS is suboptimal because it is implemented as a truncation selection, i. e. it evaluates and selects individuals instead of managing diversity at the level of the selected set. We showed how both approaches can be improved and unified by optimizing a weighted index that balances gain with a diversity measure that either minimizes inbreeding or aims to maintain rare alleles—using the flexible local searches provided by the JAMES framework. Both of these strategies yield similar results that outperform GOCS and WGS, as they provide a

better balance between genetic merit and inbreeding control, and as such between short- and long-term gain.

Although the long-term genomic selection strategies proposed in chapter 5 require further testing in other breeding schemes, we believe that their inherent characteristics will transfer from our simulations to many practical breeding settings. A possible next step could be to move from selecting individuals to selecting crosses, i. e. specific pairs of individuals to cross, as has already been successfully done for selection strategies based on the optimal contributions theory. We expect that, in particular, the advantages of the proposed strategies using the suggested alternative diversity measures will transfer to this related problem and outperform existing methods, but further research is required to validate this expectation.

For simple traits controlled by a small number of genes, such as many disease resistances, we can go much further than prediction of breeding value from genetic marker data. Because few genes are involved we can fully define the genetic profile of the breeding target at these particular loci and predesign a detailed crossing scheme to obtain this target from the available material, with minimum cost and within minimum time. Such crossing scheme tells the breeder in advance precisely which individuals to cross, and which genetic profiles to select from the produced offspring, in each generation. Because there are a huge number of possibilities, predesigning crossing schemes in such great detail is only feasible for simple traits and requires intelligent algorithms to explore the search space in order to identify optimal schemes.

In chapter 6 we introduced Gene Stacker: a flexible crossing scheme generator used to efficiently stack genes found in multiple existing varieties into a single new individual. Gene Stacker uses an exhaustive generation algorithm that iteratively combines crossing schemes to build larger schemes through additional crossings, and applies many exact and heuristic pruning criteria to reduce the number of explored schemes. The ultimate goal is to find schemes with a minimum number of generations (time) and total number of plants to grow and genotype (cost) while also taking into account several operational and crop specific constraints. However, these objectives are largely conflicting, meaning that reducing the time usually increases the cost and vice versa. Therefore, Gene Stacker constructs multiple schemes reflecting optimal tradeoffs between the different objectives.

Gene Stacker strongly relies on carefully designed heuristics to deal with complex stacking problems with up to ten or more genes. Although our experiments prove the value of the currently included heuristics, these are certainly not perfect nor complete—for ex-

ample, they are less effective for heterozygous breeding targets. Therefore, an important direction for future work on Gene Stacker is the design of improved or additional heuristics, in particular for heterozygous target profiles.

Another major future challenge is to unify Gene Stacker's approach for simple traits with genomic selection strategies for complex traits. Unfortunately it is not computationally feasible to track a complex genetic background—potentially made up of hundreds or thousands of genes—in as much detail as the few foreground markers. However, we believe that it should be possible to at least track the average genetic value of plants selected throughout the breeding scheme, using genomic prediction models, and maybe also the associated variance. Such augmented model would allow to pursue the additional objective of maximizing genetic value, while still ensuring that the desired foreground marker profile is obtained and minimizing time and cost—effectively unifying the emerging marker-assisted breeding approaches for complex and simple traits.

It is expected that the world population will approach ten billion by 2050. Feeding all these people poses a big challenge and will require to keep pushing the limits to produce more food on less land. At the same time, we need to fight climate changes due to global warming. Besides mitigation of climate change by reducing greenhouse gas emissions, for example through the use of alternative renewable energy sources, it is also very important to adapt to currently observed and predicted changes. Whether we like it or not—and we really shouldn't—scientists agree that the average global temperature will rise with at least 1.5 to 2 degrees Celsius as compared to pre-industrial times, which will have a significant impact on life and agriculture.

Marker-assisted selection is one of the major technologies with huge potential to lead to further successes in coping with this changing environment and vastly growing world population. Plant breeding used to be an art but science is now increasingly taking the guesswork out. Currently, molecular breeding techniques use genetic markers to improve selection and are being implemented around the globe in all major crops. One of the main future challenges is to move from per generation decision making towards effective computational breeding where genotypes are truly predesigned and created through detailed predefined crossing schemes, following the approach of Gene Stacker for simple traits.

We are confident that our work will be valuable for breeders and plant researchers, and in particular that it will support the pending transition from molecular towards computational breeding for sustainable crop improvement in the 21st century.

8

NEDERLANDSTALIGE SAMENVATTING

Reeds sinds de opkomst van de landbouw, meer dan tienduizend jaar geleden, selecteerden boeren jaar na jaar zaden van planten met goede eigenschappen om te zaaien in het volgende seizoen. Op die manier creëerden deze eerste plantenveredelaars—hoofdzakelijk zonder het zelf te beseffen—een waaier aan lokale variëteiten die heel sterk aangepast zijn aan hun omgeving. Na de Middeleeuwen begonnen onderzoekers de voortplantingsmechanismen van planten te ontcijferen, en slaagde men erin om manueel kruisingen uit te voeren, die een bijkomende bron van variatie boden om uit te selecteren. Niet veel later werden dan ook de eerste commerciële plantenveredelingsbedrijven opgericht die voortdurend nieuwe en betere variëteiten wisten te ontwikkelen door herhaaldelijk te kruisen en te selecteren op basis van uiterlijke kenmerken (het *fenotype*) en hun zaden verkochten aan landbouwers.

Het werk van de beroemde Gregor Mendel in de 19de eeuw zette een nieuw tijdperk in. Verschillende belangrijke ontdekkingen volgden elkaar op, waarbij de onderliggende genetica werd blootgelegd die verantwoordelijk is voor de diversiteit aan eigenschappen die we aantreffen in alle levende organismen, waaronder planten. Er werden methoden ontwikkeld om DNA-fragmenten te extraheren uit planten en dieren, om zo een zicht te krijgen op hun genetische architectuur—het zogenaamde *genotype*. Veredelaars kunnen deze genetische informatie goed gebruiken om betere beslissingen te maken, aangezien hun ultieme doel is om een maximum aan gunstige genen (of in feite gunstige varianten van genen, wat men *allelen* noemt) te verzamelen, zodat de plant zoveel mogelijk goede eigenschappen zal ontwikkelen. Voornamelijk tijdens de laatste paar tientallen jaren zijn de kosten om het genotype van planten te bepalen enorm gezakt, tot op een punt waar de vraag vooral is hoe men deze genetische data optimaal kan gebruiken in praktische veredelingsprogramma's.

In deze thesis passen we discrete optimalisatie-algoritmes toe om verschillende problemen op te lossen die gerelateerd zijn aan moderne *moleculaire* plantenveredeling, gebruik makend van genetische informatie. De eerste twee hoofdstukken schetsen een brede achtergrond van deze beide disciplines, en beschrijven belangrijke

concepten die nodig zijn om de opgeloste problemen en toegepaste technieken in de volgende hoofdstukken te begrijpen.

Vervolgens introduceert hoofdstuk 3 het JAMES framework, een objectgeoriënteerd Java framework voor discrete optimalisatie met lokale zoekmethoden. Dergelijke benaderingsalgoritmes verkennen de zoekruimte van een gegeven optimalisatieprobleem door te starten van een bepaalde (bijvoorbeeld random gekozen) oplossing en deze herhaaldelijk lichtjes aan te passen, in de hoop een eindresultaat te bekomen dat dicht aanleunt bij het optimum, d.w.z. bij de best mogelijke oplossing. Het JAMES framework wordt in de volgende hoofdstukken gebruikt om verschillende problemen op te lossen met dezelfde optimalisatietechnieken.

Aangezien veel van de biodiversiteit die voorkomt in geteelde gewassen niet rechtstreeks gebruikt wordt voor landbouwdoeleinden, berusten veredelaars vaak op de beschikbaarheid van de diversiteit aan zaden die wereldwijd opgeslagen wordt in grote zaadbanken. Naast moderne producten die door veredelaars gecreëerd werden, houdt men hier ook zaden bij van bijvoorbeeld wilde verwante planten en historische lokale variëteiten. Deze vormen samen een onmisbare bron van diversiteit voor veredelaars en plantenonderzoekers in het algemeen. Ondertussen zijn de opgeslagen collecties echter zo groot geworden dat het moeilijk wordt om ze in hun geheel gedetailleerd te karakteriseren, en om gebruikers toegang te verlenen tot de volledige collecties. Daarom stelt men vaak kleinere zogenaamde *core collecties* samen, die de diversiteit van de volledige verzameling weerspiegelen met zo weinig mogelijk overtolligheid.

In hoofdstuk 4 introduceren we Core Hunter 3: een flexibel softwareprogramma voor het samenstellen van core collecties voor verscheidene doeleinden. Afhankelijk van de toepassing kan men met Core Hunter de diversiteit binnen de geselecteerde deelverzameling maximaliseren, of de mate waarin elke individuele plant uit de volledige collectie weerspiegeld wordt door de selectie, of ten slotte de rijkdom aan bewaarde allelen. Core Hunter maakt gebruik van lokale zoekstrategieën uit het JAMES framework om de gekozen doelfunctie, of een gewogen combinatie van meerdere objectieven, te optimaliseren. Een van de voornaamste voordelen van Core Hunter is de inherente flexibiliteit. Zo kunnen bijvoorbeeld eenvoudig nieuwe criteria toegevoegd worden, zonder de gebruikte optimalisatie-algoritmes aan te moeten passen. Bovendien presteert Core Hunter even goed of beter dan andere methodes die specifiek ontwikkeld werden voor bepaalde toepassingen van core collecties.

Van zodra het startmateriaal verzameld is, gaan plantenveredelaars typisch door meerdere generaties van herhaalde kruising en selectie om gunstige genen van verschillende bronnen op te stapelen in een

nieuw, verbeterd product. Jammer genoeg hebben kwantitatieve eigenschappen—zoals hoogte, grootte of opbrengst—doorgaans een complexe genetische architectuur waarbij veel genen verspreid over het genoom elk een klein additief effect hebben op de kwaliteit van de plant. Wetenschappers zijn daarom druk in de weer om alle genen te identificeren die een effect hebben op essentiële eigenschappen in de belangrijkste gewassen. Indien al deze genen gekend zouden zijn, kan men zich er immers op toespitsen om deze allemaal te verzamelen in een enkel genotype om zo het ontwikkelde fenotype maximaal te verbeteren. Helaas zijn er zoveel genen betrokken bij deze complexe eigenschappen dat het zeer moeilijk is om ze allemaal te lokaliseren.

Een praktische oplossing is om het genotype van een plant te karakteriseren aan de hand van een grote hoeveelheid zogenaamde genetische merkers verspreid over het hele genoom. Dergelijke merkers zijn kleine fragmentjes DNA die men gemakkelijk kan opsporen en die variëren tussen individuen. Op basis van de merkers die we aantreffen in een bepaalde plant kan de kwaliteit van deze plant dan voorspeld worden. Hiervoor traint men een predictiemodel op een verzameling planten waarvoor zowel het genetisch profiel (de gedetecteerde merkers) en de kwaliteit gekend is. In de hoop dat het effect van de meeste causale genen opgepikt wordt door een nabijgelegen merker, kan zo'n predictiemodel vrij nauwkeurig de kwaliteit voorspellen van andere genetische profielen, zonder te moeten weten waar de causale genen precies gelegen zijn.

Selecteren op basis van dergelijke voorspelde kwaliteit wordt *genomische selectie* genoemd en is een van de belangrijkste trends bij de opkomst van moleculaire veredeling doormiddel van *merkergebaseerde* selectietechnieken voor complexe eigenschappen. Een van de grote voordelen is dat de selectiecyclus versneld wordt—de dure en tijdrovende stap voor het vaststellen van de uiterlijke kenmerken van (vaak volwassen) planten kan namelijk overgeslagen of op zijn minst uitgesteld worden—terwijl tegelijkertijd ook de nauwkeurigheid van de selectie verbetert, voornamelijk voor kenmerken die moeilijk te observeren zijn. Helaas is het ook reeds geweten dat genomische selectie ervoor zorgt dat de diversiteit in de populatie sneller uitgeput raakt. Zonder diversiteit om uit te selecteren is er geen vooruitgang meer mogelijk. Bijgevolg kan het gebruik van genomische selectie de vooruitgang op lange termijn belemmeren, niettegenstaande dat er op korte termijn doorgaans sneller meer winst geboekt wordt.

In hoofdstuk 5 bespreken we enkele bestaande en nieuwe strategieën voor genomische selectie op lange termijn, die onmiddellijke vooruitgang afwegen tegenover het behoud van diversiteit. Een

eerste bestaande methode maximaliseert vooruitgang onder een beperkte, vooropgestelde mate van inteelt. Een tweede alternatief versterkt artificieel het geschatte effect van zeldzame gunstige allelen in het predictiemodel, om te voorkomen dat deze tijdens de selectie verloren gaan. Onze simulaties tonen aan dat beide methoden intrinsieke beperkingen hebben: de eerste strategie slaagt er niet in om de inteelt effectief te beperken tot het gewenste niveau, omdat ze een belangrijke component negeert die enkel optreedt onder selectie, en de tweede aanpak is suboptimaal omdat ze werkt op het niveau van de individuen in plaats van diversiteit rechtstreeks te beheren voor de gekozen verzameling. Wij tonen hoe beide strategieën verbeterd en verenigd kunnen worden door een gewogen index te optimaliseren die vooruitgang balanceert met een diversiteitsmaat die ofwel inteelt minimaliseert, of ernaar streeft om zeldzame allelen te behouden. Opnieuw maken we hiervoor gebruik van de flexibele lokale zoekalgoritmes die voorzien zijn in het JAMES framework. Beide nieuwe selectiemethodes presteren in onze simulaties gelijkaardig, en beter dan de bestaande technieken, aangezien ze een betere balans vinden tussen onmiddellijke winst en behoud van diversiteit, d.w.z. een betere balans tussen vooruitgang op korte en lange termijn.

Voor eenvoudige kenmerken die bepaald worden door een klein aantal genen, zoals bijvoorbeeld heel wat ziekteresistenties, kunnen we veel verder gaan dan het voorspellen van kwaliteit op basis van genetische merkers. In dat geval is het namelijk mogelijk om het genetisch profiel dat we willen bekomen volledig te definiëren voor de weinige betrokken genen. Bijgevolg kunnen we op voorhand een gedetailleerd kruisingsschema opstellen dat dit doel zo goedkoop mogelijk en binnen zo weinig mogelijk tijd bereikt. Een dergelijk kruisingsschema vertelt de veredelaar op voorhand precies welke individuen gekruist moeten worden, en welke genetische profielen geselecteerd moeten worden uit de nakomelingen, in elke generatie. Omdat er ongelofelijk veel mogelijkheden zijn, is het enkel voor eenvoudige kenmerken doenbaar om kruisingsschema's in zo'n detail op te stellen, en hebben we intelligente algoritmes nodig om de zoekruimte te verkennen en optimale schema's te identificeren.

In hoofdstuk 6 introduceren we Gene Stacker: een flexibele generator van kruisingsschema's om efficiënt genen uit verschillende bestaande individuen te verzamelen in een enkele nieuwe plant. Gene Stacker gebruikt een exhaustief generatie-algoritme dat iteratief kruisingsschema's combineert om grotere schema's te bouwen door extra kruisingen toe te voegen. Om het aantal geconstrueerde schema's enigszins te beperken worden hierbij verschillende exacte en benaderende (zogenaamde *heuristische*) snoeicriteria gebruikt, die delen van de zoekruimte overslaan. Het hoofddoel is

om schema's te vinden met een minimum aantal generaties (minimale tijd) en om zo weinig mogelijk planten te moeten kweken en screenen (minimale kost) terwijl ook rekening gehouden dient te worden met enkele operationele en plantspecifieke beperkingen, zoals het maximaal aantal planten dat men in een generatie kan kweken en het aantal zaden dat een kruising normaal gezien oplevert. De twee hoofddoelen zijn echter grotendeels conflicterend, wat betekent dat schema's met minder generaties doorgaans een hogere kost hebben en omgekeerd. Daarom construeert Gene Stacker meerdere schema's die de verschillende doelen optimaal tegenover elkaar afwegen. De kracht van Gene Stacker berust voor een groot deel op de vele doordachte *heuristieken*. Deze slaan delen van de zoekruimte over wanneer het onwaarschijnlijk is dat daar een goede oplossing gevonden zal worden, met als doel om zoveel mogelijk tijd te winnen en zo weinig mogelijk in te boeten in de kwaliteit van de gevonden oplossingen. Dit resulteert in een interessante afweging tussen uitvoeringstijd en kwaliteit van de voorgestelde schema's, die het toelaat om complexe problemen met tien of meer genen aan te pakken, zoals blijkt uit onze experimenten.

Naar verwachting zal de wereldpopulatie in 2050 tegen de tien miljard aanleunen. De grote uitdaging om al deze mensen van voedsel te voorzien zal ervoor blijven zorgen dat men steeds meer gewassen moet produceren op minder land. Tegelijkertijd moeten we strijden tegen klimaatverandering als gevolg van de opwarming van de aarde. Bovenop initiatieven om verdere opwarming tegen te gaan, door bijvoorbeeld de uitstoot van broeikasgassen te verminderen, moeten we ons ook weten aan te passen aan de reeds vastgestelde en voorspelde veranderingen. We mogen ons niet laten vangen door wat sommige bronnen durven te beweren of te ontkennen: wetenschappers zijn het er algemeen over eens dat de gemiddelde globale temperatuur met minstens 1.5 tot 2 graden Celsius zal stijgen in vergelijking met preïndustriële tijden, wat een significante impact zal hebben op ons leven en op de landbouw in het bijzonder, en de mensheid heeft hier een duidelijke rol in gespeeld.

Merker-gebaseerde selectie is een van de belangrijkste technologieën met een zeer groot potentieel om tot verdere successen te leiden bij onze opdracht om de snelgroeiende wereldbevolking van voedsel te blijven voorzien, in tijden van klimaatverandering. Gedurende lange tijd was plantenveredeling voornamelijk een kunst, maar dankzij de wetenschap is het giswerk meer en meer aan het verdwijnen. Op dit moment worden steeds meer moderne moleculaire technieken toegepast die genetische informatie gebruiken om betere selecties te maken. Een van de grootste uitdagingen voor de toekomst is om over te gaan van generatie-per-generatie beslissingen naar effectieve computationele veredeling, waarbij nieuwe

genetische configuraties werkelijk in detail ontworpen zijn en vervolgens gecreëerd worden via specifieke op voorhand gedefinieerde kruisingsschema's, zoals Gene Stacker momenteel doet voor eenvoudige kenmerken en de enkele betrokken genen.

We hebben er vertrouwen in dat ons werk waardevol zal zijn voor veredelaars en plantenonderzoekers, en dat het in het bijzonder zal bijdragen aan de aanstaande overgang van moleculaire naar computationele veredeling, met het oog op duurzame ontwikkeling van verbeterde gewassen in de 21e eeuw.

Part V

APPENDIX

A

LONG-TERM GS: SUPPLEMENTARY MATERIAL

A.1 DATA PREPROCESSING DETAILS

The founder dataset, consisting of 192 individuals and 2591 SNPs, was cleaned and preprocessed as follows:

1. Remove markers (223) with unknown position in the genetic map at hand.
2. Remove markers (46) and individuals (0) with more than 20% missing values.
3. Recode markers as number of copies of minor allele (0,1,2).
4. Impute missing values using Beagle 3.3.2 (Browning and Browning, 2009; Browning and Browning, 2007) through the R package synbreed (version 0.10-5) (Wimmer et al., 2015). Beagle uses a hidden Markov model (HMM) to reconstruct missing values based on flanking markers.
5. Filter redundant markers (291). Markers were considered redundant if they had the same map position and the same allele was observed in all individuals. From each set of redundant markers, only one was retained.
6. Spread remaining markers mapped to same position at 0.1 cM intervals in arbitrary order.

This procedure retained 2031 polymorphic SNPs and all 192 individuals.

A.2 GENOMIC OPTIMAL CONTRIBUTIONS SELECTION

A.2.1 *Genomic inbreeding control*

Here, we formally derive how the inbreeding rate ΔF relates to SNP allele frequencies in the population and the changes of these frequencies over time, and to the GOCS constraint C_{t+1}. First we

express C_{t+1} in terms of SNP allele frequencies starting from its definition

$$C_{t+1} = \frac{\mathbf{c}_t^\intercal \mathbf{G}_t \mathbf{c}_t}{2}.$$

Here, $\mathbf{c}_t$ is a vector of assigned contributions (under optimization) and $\mathbf{G}_t$ is the realized genomic relationship matrix of the selection candidates in generation t, defined as

$$\mathbf{G}_t = \frac{\mathbf{Z}_t \mathbf{Z}_t^\intercal}{2\sum_{j=1}^{m} p_j(1-p_j)}$$

where m is the number of markers, p_j is the reference allele frequency of the j-th marker in the population of selection candidates and $\mathbf{Z}_t$ is the centered marker matrix of the selection candidates:

$$\mathbf{Z}_t = \mathbf{X}_t - 2\mathbf{P}_t$$

where $\mathbf{X}_t$ is the original marker matrix containing reference allele counts (0/1/2) and $\mathbf{P}_t$ is a matrix whose j-th column contains the current allele frequency p_j of the j-th marker:

$$\mathbf{P}_t = \begin{bmatrix} p_1 & p_2 & \cdots & p_m \\ p_1 & p_2 & \cdots & p_m \\ \vdots & \vdots & \ddots & \vdots \\ p_1 & p_2 & \cdots & p_m \end{bmatrix}.$$

As such, the values of $\mathbf{Z}_t$ represent reference allele counts relative to the population mean and each of its columns sums to zero. It follows that (Woolliams et al., 2015)

$$\mathbf{Z}_t^\intercal \mathbf{c}_t = 2 \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_m \end{bmatrix}$$

where Δ_j is the expected allele frequency change when mating the individuals from the selection population according to the assigned contributions $\mathbf{c}_t$. Therefore

$$\mathbf{c}_t^\intercal \mathbf{Z}_t \mathbf{Z}_t^\intercal \mathbf{c}_t = 4\sum_{j=1}^{m} \Delta_j^2$$

and thus

$$C_{t+1} = \frac{\mathbf{c}_t^\intercal \mathbf{G}_t \mathbf{c}_t}{2} = \frac{\mathbf{c}_t^\intercal \mathbf{Z}_t \mathbf{Z}_t^\intercal \mathbf{c}_t}{4\sum_{j=1}^{m} p_j(1-p_j)} = \frac{1}{2m}\frac{\mathbf{c}_t^\intercal \mathbf{Z}_t \mathbf{Z}_t^\intercal \mathbf{c}_t}{H_t} = \frac{2}{mH_t}\sum_{j=1}^{m} \Delta_j^2$$

where $H_t = \frac{1}{m}\sum_{j=1}^{m} 2p_j(1-p_j)$ is the expected heterozygosity in the selection population.

Now we also express the inbreeding rate ΔF in terms of the SNP allele frequencies and their changes, starting from its definition:

$$\Delta F = \frac{F_{t+1} - F_t}{1 - F_t}$$

with (for the SNP marker panel used)

$$F_t = \frac{1}{m}\sum_{j=1}^{m} p_j^2 + (1-p_j)^2 \quad \text{and} \quad 1 - F_t = H_t.$$

It follows that

$$\begin{aligned}
\Delta F_{IBS} &= \frac{1}{mH_t}\left[\sum_{j=1}^{m}(p_j+\Delta_j)^2 + (1-p_j-\Delta_j)^2 - \sum_{j=1}^{m} p_j^2 + (1-p_j)^2\right] \\
&= \frac{1}{mH_t}\sum_{j=1}^{m} p_j^2 + 2p_j\Delta_j + \Delta_j^2 + (1-p_j)^2 - 2(1-p_j)\Delta_j + \Delta_j^2 - p_j^2 - (1-p_j)^2 \\
&= \frac{1}{mH_t}\sum_{j=1}^{m} 2\Delta_j^2 + 2p_j\Delta_j - 2(1-p_j)\Delta_j \\
&= \frac{1}{mH_t}\sum_{j=1}^{m} 2\Delta_j^2 + 4p_j\Delta_j - 2\Delta_j \\
&= \frac{2}{mH_t}\sum_{j=1}^{m}\Delta_j^2 + \frac{2}{mH_t}\sum_{j=1}^{m}\Delta_j(2p_j-1) \\
&= \frac{\mathbf{c_t^T G_t c_t}}{2} + \frac{2}{mH_t}\sum_{j=1}^{m}\Delta_j(2p_j-1) \\
&= C_{t+1} + \frac{2}{mH_t}\sum_{j=1}^{m}\Delta_j(2p_j-1).
\end{aligned}$$

As such, the inbreeding rate ΔF_{IBS} defined for SNP markers is the sum of the GOCS constraint $C_{t+1} = \mathbf{c_t^T G_t c_t}/2$ and an additional term $\frac{2}{mH_t}\sum_{j=1}^{m}\Delta_j(2p_j-1)$ that is not constrained by GOCS.

A.2.2 *Selection procedure*

The optimal contributions selection (OCS) strategy was originally presented in an animal breeding context by Meuwissen (1997) where it is required that the contributions of males and females both sum to 1/2. In our plant breeding scheme such constraint

does not apply and all contributions must simply sum to one, i.e. $\sum \mathbf{c_t} = 1$. This slightly changes the optimization formulas of GOCS based on Lagrangian multipliers to

$$\mathbf{c_t} = \frac{\mathbf{G_t^{-1}}(\mathbf{GEBV_t} - \lambda)}{2\lambda_0}$$

$$\lambda = \frac{\mathbf{1}^\intercal \mathbf{G_t^{-1}} \mathbf{GEBV_t} - 2\lambda_0}{\mathbf{1}^\intercal \mathbf{G_t^{-1}} \mathbf{1}}$$

$$\lambda_0^2 = \frac{\mathbf{GEBV_t}^\intercal (\mathbf{G_t^{-1}} - \frac{\mathbf{G_t^{-1}} \mathbf{1}\mathbf{1}^\intercal \mathbf{G_t^{-1}}}{\mathbf{1}^\intercal \mathbf{G_t^{-1}} \mathbf{1}}) \mathbf{GEBV_t}}{8C_{t+1} - \frac{4}{\mathbf{1}^\intercal \mathbf{G_t^{-1}} \mathbf{1}}}.$$

Any negative contributions are eliminated by setting the most negative value to zero and iteratively re-optimizing the remaining contributions. Following Meuwissen (2002) a minimum and/or maximum contribution, c_{min} and c_{max}, respectively, may be imposed. As a special case we set $c_{min} = c_{max} = 1/n$ to select n individuals with equal contribution. To deal with these additional constraints, contributions exceeding the maximum value are truncated and those individuals with a too low contribution are discarded. In each step of the algorithm, the following rules are applied to adjust the contributions, after which the remaining ones are re-optimized:

1. Discard the individual with the most negative contribution, if any, by fixing its contribution to zero.
2. Else, if any contribution exceeds the imposed maximum c_{max}, truncate the largest contribution to c_{max} and exclude the corresponding individual from the optimization. In addition, all individuals that were previously discarded, if any, are re-included in the optimization.
3. Else, if any selected individual has a contribution below the imposed minimum c_{min}, discard the individual with the smallest positive contribution.

Meuwissen (1997) explained in an appendix how to extend the formulas to optimize the remaining contributions $\mathbf{c_o}$ when some

have already been fixed to $\mathbf{c_f}$, in our case to either zero or $c_{max} = 1/n$. Adjusting the formulas for our plant breeding scheme yields

$$\mathbf{c_o} = \frac{\mathbf{G_{oo}^{-1}}(\mathbf{GEBV_o} - 2\lambda_0 \mathbf{G_{of}} \mathbf{c_f} - \lambda)}{2\lambda_0}$$

$$\lambda = \frac{\mathbf{1}^\intercal \mathbf{G_{oo}^{-1}}(\mathbf{GEBV_o} - 2\lambda_0 \mathbf{G_{of}} \mathbf{c_f}) - 2\lambda_0 s}{\mathbf{1}^\intercal \mathbf{G_{oo}^{-1}} \mathbf{1}}$$

$$\lambda_0^2 = \frac{1}{4} \frac{\mathbf{GEBV_o}^\intercal \, \mathbf{P} \, \mathbf{GEBV_o}}{K + L - M - N}$$

where

$$s = 1 - \sum c_f$$

$$\mathbf{P} = \mathbf{G_{oo}^{-1}} - \frac{\mathbf{G_{oo}^{-1}} \mathbf{1}\mathbf{1}^\intercal \mathbf{G_{oo}^{-1}}}{\mathbf{1}^\intercal \mathbf{G_{oo}^{-1}} \mathbf{1}}$$

$$K = 2C_{t+1} - \mathbf{c_f}^\intercal \mathbf{G_{ff}} \mathbf{c_f}$$

$$L = \mathbf{c_f}^\intercal \mathbf{G_{fo}} \mathbf{P} \mathbf{G_{of}} \mathbf{c_f}$$

$$M = \frac{s^2}{\mathbf{1}^\intercal \mathbf{G_{oo}^{-1}} \mathbf{1}}$$

$$N = \frac{2s \mathbf{1}^\intercal \mathbf{G_{oo}^{-1}} \mathbf{G_{of}} \mathbf{c_f}}{\mathbf{1}^\intercal \mathbf{G_{oo}^{-1}} \mathbf{1}}.$$

Here, $\mathbf{GEBV_o}$ is a vector of genomic estimated breeding values of the individuals under optimization, and $\mathbf{G_{xy}}$ is the genomic relationship matrix restricted to rows $\mathbf{x}$ and columns $\mathbf{y}$. Applying these formulas optimizes the remaining contributions $\mathbf{c_o}$ to maximize the expected genetic gain $\mathbf{c_o}^\intercal \mathbf{GEBV_o}$ while constraining

$$C_{t+1} = \frac{\mathbf{c_t}^\intercal \mathbf{G_t} \mathbf{c_t}}{2} = \frac{1}{2}[\mathbf{c_o}^\intercal \mathbf{G_{oo}} \mathbf{c_o} + 2\mathbf{c_o}^\intercal \mathbf{G_{of}} \mathbf{c_f} + \mathbf{c_f}^\intercal \mathbf{G_{ff}} \mathbf{c_f}]$$

to the target inbreeding rate $C_{t+1} = \Delta F_{target}$, with

$$\sum \mathbf{c_t} = \sum (\mathbf{c_o} + \mathbf{c_f}) = 1.$$

It may happen that, in a certain step of the iterative heuristic, this constraint cannot be satisfied for the remaining individuals. In such case, we assign the remaining contributions by minimizing the corresponding realized genomic relationship, in order to approach the requested constraint value as closely as possible. Formulas for

Table A.1: Used R packages and versions.

Package	Version	Reference(s)
BGLR	1.0.4	Campos and Pérez (2015)
coda	0.17-1	Plummer et al. (2006)
rrBLUP	4.3	Endelman (2011)
synbreed	0.10-5	Wimmer et al. (2015)
hypred	0.5	Technow (2014)
gdata	2.17.0	Warnes et al. (2015)
Hmisc	3.16-0	Harrell et al. (2015)
rJava	0.9-7	Urbanek (2015)
setRNG	2013.9-1	Gilbert (2014)

the latter optimization problem are also obtained with Lagrangian multipliers by minimizing $\mathbf{c_t}^\intercal \mathbf{G_t} \mathbf{c_t}/2$ with $\sum \mathbf{c_t} = 1$:

$$\mathbf{c_o} = \mathbf{G_{oo}^{-1}}(\frac{1}{2}\lambda - \mathbf{G_{of}}\mathbf{c_f})$$

$$\lambda = \frac{2(1 - \sum \mathbf{c_f} + \mathbf{1}^\intercal \mathbf{G_{oo}^{-1}} \mathbf{G_{of}} \mathbf{c_f})}{\mathbf{1}^\intercal \mathbf{G_{oo}^{-1}} \mathbf{1}}.$$

A.3 SUPPLEMENTARY FIGURES AND TABLES

Table A.1 lists all R packages that were used for this study and figures A.1 to A.5 provide results for additional simulation settings.

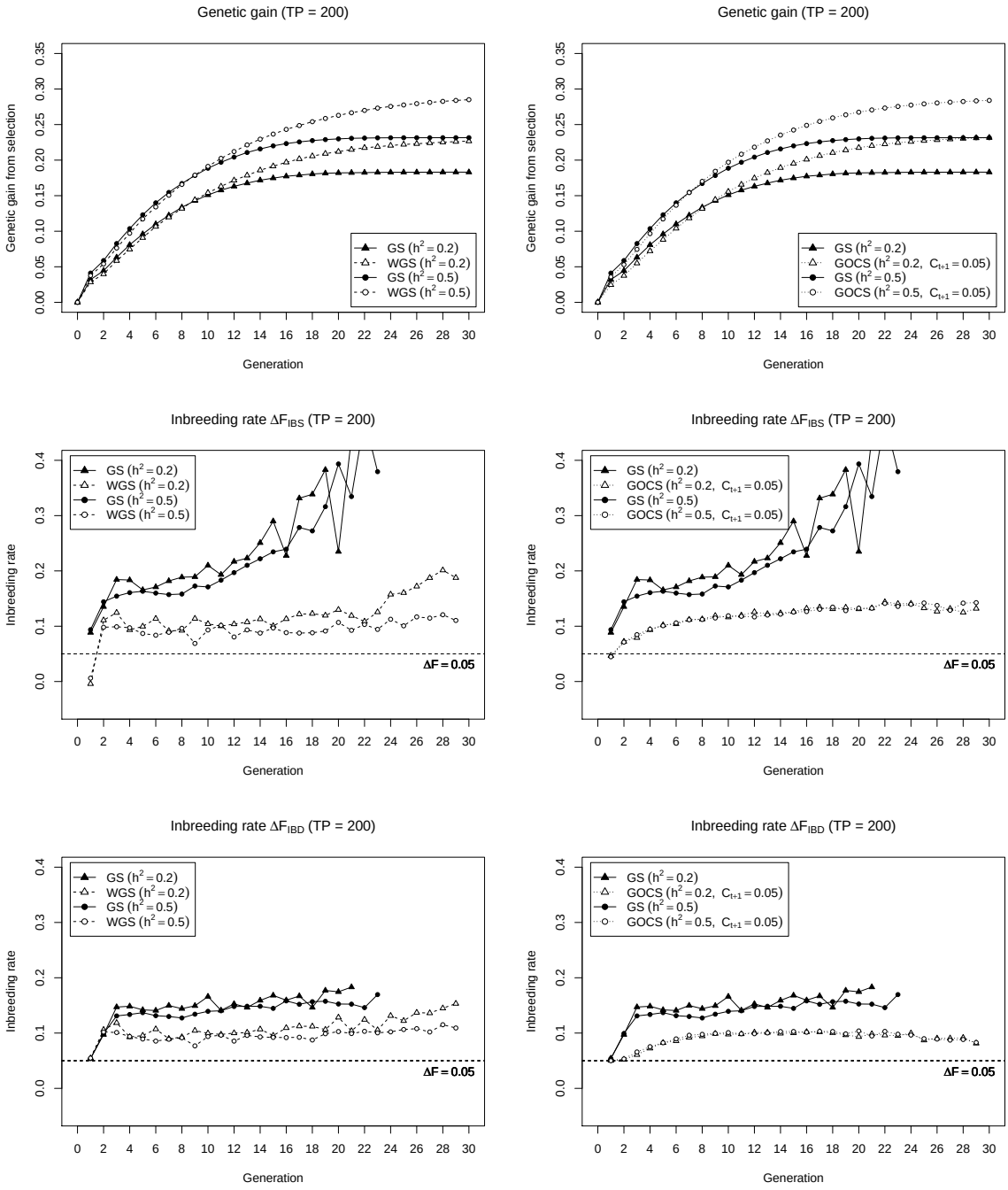


Figure A.1: Cumulative genetic gain (top) and inbreeding rate (IBS: middle; IBD: bottom) for weighted genomic selection (WGS; left) and genomic optimal contributions selection (GOCS; right) as compared to standard genomic selection (GS). Results are reported for a low ($h^2 = 0.2$) and high ($h^2 = 0.5$) heritability with a small initial training population (TP = 200) and are averages of 200 simulation runs. The inbreeding rates are reported until at least half of the simulation runs have lost all variability for the SNP marker panel used.

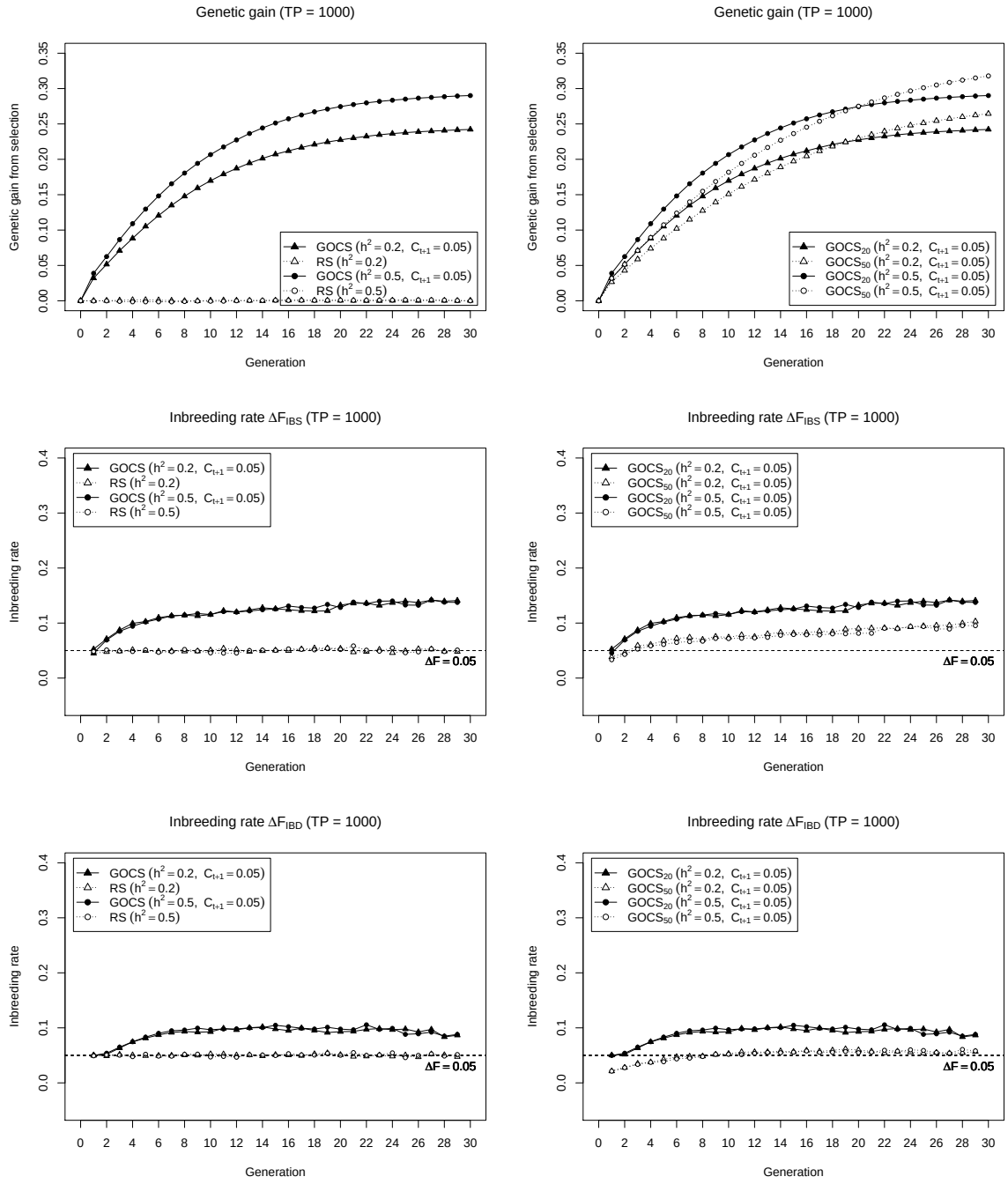


Figure A.2: Cumulative genetic gain (top) and inbreeding rate (IBS: middle; IBD: bottom) when no selection is performed (RS; left), i.e. where 20 individuals are chosen randomly in each cycle, and for genomic optimal contributions selection with a larger selection consisting of 50 individuals ($GOCS_{50}$; right), as compared to GOCS with the default selection size (20). Results are reported for a low ($h^2 = 0.2$) and high ($h^2 = 0.5$) heritability with a large initial training population (TP = 1000) and are averages of 200 simulation runs.

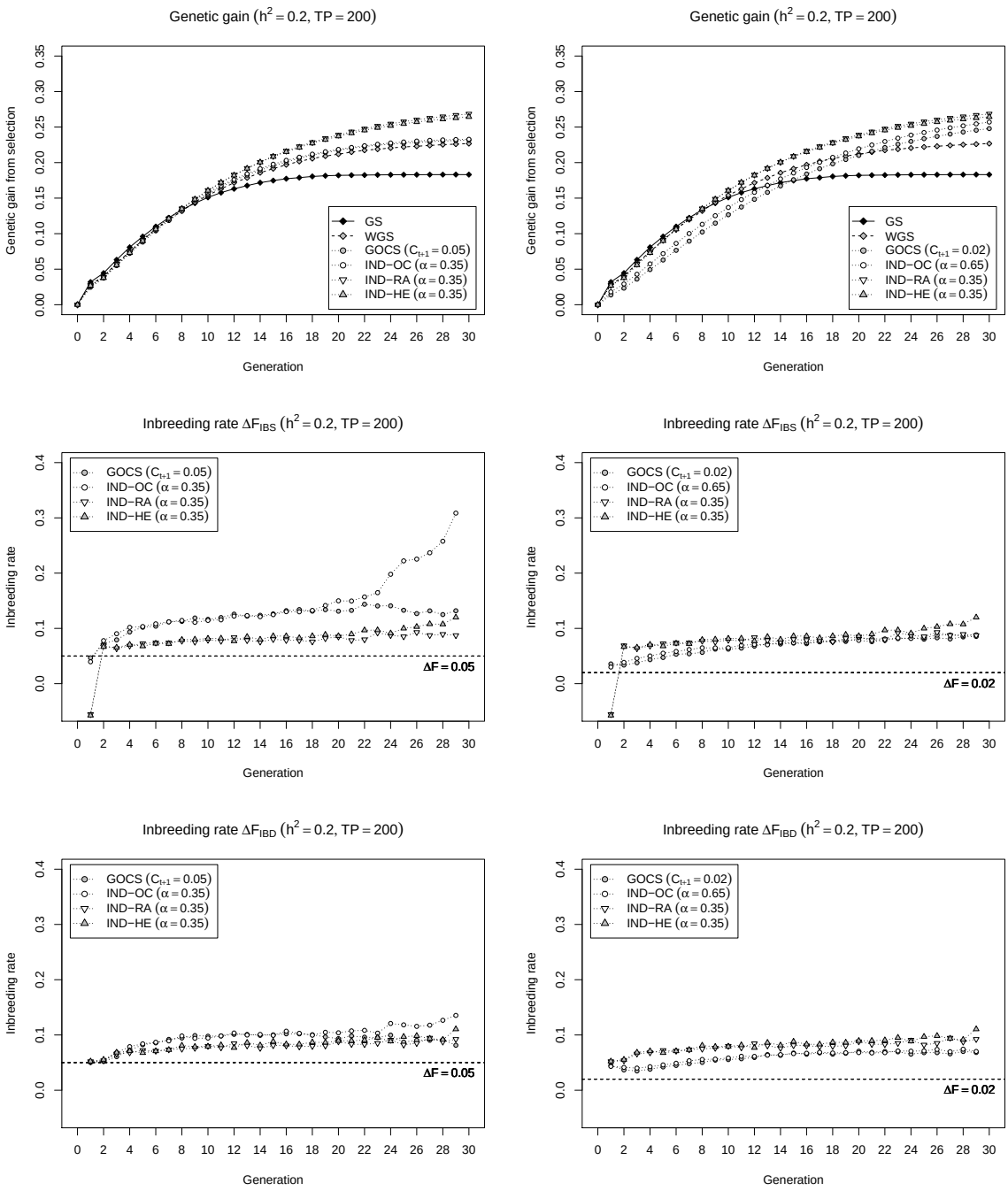


Figure A.3: Cumulative genetic gain (top) and inbreeding rate (IBS: middle; IBD: bottom) of selection strategies that maximize a weighted index containing breeding value and a diversity measure chosen to control inbreeding (IND-OC, IND-HE) or to avoid loss of rare alleles (IND-RA). Results for GS, WGS, and GOCS are provided as a reference. For clarity, inbreeding rates of GS and WGS are omitted. Two scenarios were considered to set the parameters C_{t+1} and α: maintain the same short-term gain as WGS (left), or achieve a similar inbreeding rate ΔF_{IBS} (right). Results are reported for a low heritability ($h^2 = 0.2$) with a small initial training population (TP = 200) and are averages of 200 simulation runs.

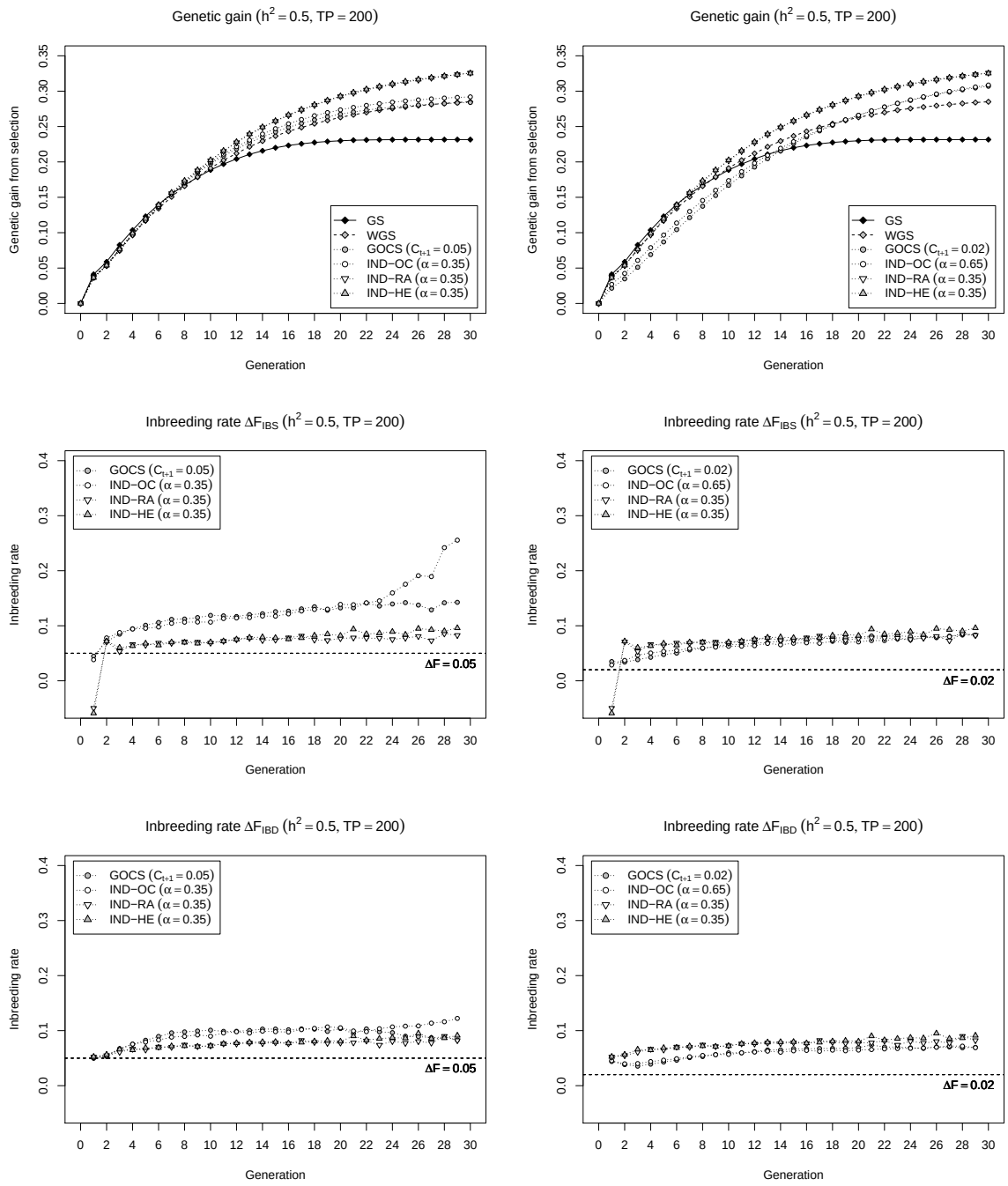


Figure A.4: Cumulative genetic gain (top) and inbreeding rate (IBS: middle; IBD: bottom) of selection strategies that maximize a weighted index containing breeding value and a diversity measure chosen to control inbreeding (IND-OC, IND-HE) or to avoid loss of rare alleles (IND-RA). Results for GS, WGS, and GOCS are provided as a reference. For clarity, inbreeding rates of GS and WGS are omitted. Two scenarios were considered to set the parameters C_{t+1} and α: maintain the same short-term gain as WGS (left), or achieve a similar inbreeding rate ΔF_{IBS} (right). Results are reported for a high heritability ($h^2 = 0.5$) with a small initial training population (TP = 200) and are averages of 200 simulation runs.

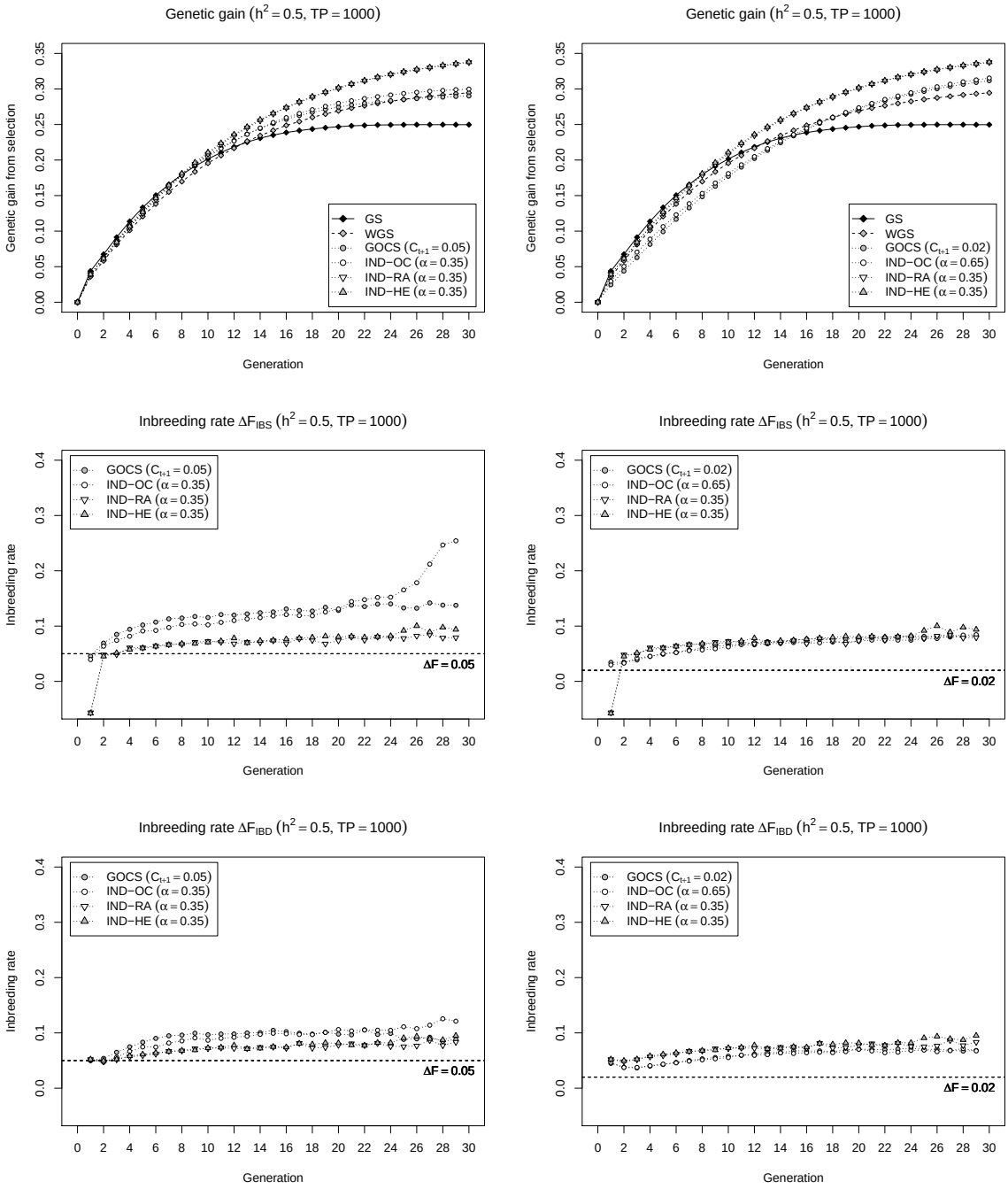


Figure A.5: Cumulative genetic gain (top) and inbreeding rate (IBS: middle; IBD: bottom) of selection strategies that maximize a weighted index containing breeding value and a diversity measure chosen to control inbreeding (IND-OC, IND-HE) or to avoid loss of rare alleles (IND-RA). Results for GS, WGS, and GOCS are provided as a reference. For clarity, inbreeding rates of GS and WGS are omitted. Two scenarios were considered to set the parameters C_{t+1} and α: maintain the same short-term gain as WGS (left), or achieve a similar inbreeding rate ΔF_{IBS} (right). Results are reported for a high heritability ($h^2 = 0.5$) with a large initial training population (TP = 1000) and are averages of 200 simulation runs.

B

GENE STACKER: FORMULAS, FIGURES & DATA

B.1 DISTRIBUTION OF GENERATED OFFSPRING

When crossing two genotypes P and Q a number of possible offspring can be produced due to crossover events, each with a certain probability. Here we describe how to compute these probabilities from the recombination rates $r_{i,p,q}$ between the p-th and q-th loci of the it-h chromosome, as inferred from the genetic map.

Take the i-th chromosome P_i of genotype P and any haplotype H_i with the same number of loci as P_i. Suppose that P_i contains l heterozygous loci with ordered indices $s = (\nu_1, \ldots, \nu_l)$. Then, the probability $Pr[P_i \to H_i]$ that haplotype H_i is produced from chromosome P_i is computed following Canzar and El-Kebir (2011):

- If H_i contains at least one allele which does not occur at the respective locus in P_i then $Pr[P_i \to H_i] = 0$, i. e. it is impossible that H_i has been produced from P_i.
- Else, if all loci are homozygous (s is empty): $Pr[P_i \to H_i] = 1$.
- Else

$$Pr[P_i \to H_i] = \frac{1}{2} \prod_{j=1}^{l-1} \begin{cases} r_{i,\nu_j,\nu_{j+1}} & \text{in case of a crossover} \\ & \text{between loci } \nu_j \text{ and } \nu_{j+1} \\ 1 - r_{i,\nu_j,\nu_{j+1}} & \text{otherwise} \end{cases}$$

  where there has been a crossover between loci ν_j and ν_{j+1} if

$$\begin{aligned} H_i(\nu_j) = P_{i,1}(\nu_j) \quad \wedge \quad H_i(\nu_{j+1}) = P_{i,2}(\nu_{j+1}), \text{ or} \\ H_i(\nu_j) = P_{i,2}(\nu_j) \quad \wedge \quad H_i(\nu_{j+1}) = P_{i,1}(\nu_{j+1}). \end{aligned}$$

  The factor of $1/2$ is introduced because every sequence of crossovers defines two complementary haplotypes which are inherited with equal probability.

Now, take any chromosome G_i with the same number of loci as the i-th chromosomes P_i and Q_i of the two parents P and Q, respectively. The probability $Pr[P_i, Q_i \to G_i]$ that chromosomes P_i and Q_i

will produce haplotypes which together form a chromosome G_i is computed as follows:

$$\Pr[P_i, Q_i \to G_i] = \begin{cases} \Pr[P_i \to G_{i,1}] \cdot \Pr[Q_i \to G_{i,2}] & \text{if } G_{i,1} = G_{i,2} \\ \Pr[P_i \to G_{i,1}] \cdot \Pr[Q_i \to G_{i,2}] & \text{if } G_{i,1} \neq G_{i,2} \\ \quad + \Pr[P_i \to G_{i,2}] \cdot \Pr[Q_i \to G_{i,1}]. & \end{cases}$$

The second case accounts for the fact that the haplotypes might swap their originating parents. As Gene Stacker explicitly models multiple chromosomes, the probability $\Pr[P, Q \to G]$ of obtaining the entire phase-known genotype G from crossing the phase-known parents P and Q, with k chromosomes, is computed by multiplying the independent chromosome probabilities:

$$\Pr[P, Q \to G] = \prod_{i=1}^{k} \Pr[P_i, Q_i \to G_i].$$

Note that this will account for up to 2^k identical phase-known genotypes G depending on how many haplotype pairs might swap their originating parents.

B.2 JOINT POPULATION SIZES

Sometimes several different genotypes or multiple occurrences of the same genotype are simultaneously targeted among offspring grown from the same seed lot. In such case it is possible to compute a joint population size, i. e. the number of offspring that needs to be generated so that at least the number of desired occurrences of each targeted genotype are expected to be obtained. The same individual and overall success rates are still guaranteed, but the computed joint population size is often much smaller than the sum of the number of offspring required to obtain each targeted genotype individually. This procedure may therefore significantly reduce the total population size.

Suppose that m distinct phase-known genotypes $G_1, \ldots, G_m$ are targeted among the offspring, where $f_1, \ldots, f_m$ occurrences of the respective targets are desired, with $f_i \geqslant 1, \forall i = 1, \ldots, m$ and $\sum_{i=1}^{m} f_i = f$. The joint population size N is then calculated as follows:

1. Compute the population sizes N_i required to obtain each target G_i individually using equation (6.1).
2. Set
$$N = \max\{N_i | i = 1, \ldots, m\}$$

and compute the joint probability of success (see below)

$$P = Pr[|G_1| \geqslant f_1 \;\&\; \cdots \;\&\; |G_m| \geqslant f_m; N].$$

3. If $P < (\gamma')^f$, adjust N with a binary search in the interval

$$I = [\max\{N_i | i = 1, \ldots, m\}, \sum_{i=1}^{m} (f_i \cdot N_i)]$$

to find the smallest $N \in I$ for which $P \geqslant (\gamma')^f$.

Taking the maximum in step 2 ensures a success rate of at least γ' for every individual target. If needed, step 3 further increases this initial estimate of the joint population size to guarantee a joint success rate of at least $(\gamma')^f$ so that this joint trial with f targets still contributes a factor of $(\gamma')^f$ to the overall success rate—just as if f independent Bernoulli trials would have been performed. Such independent trials would require a population size of $\sum_{i=1}^{m} (f_i \cdot N_i)$ to obtain all targets, which is the upper bound of the interval I. By considering a joint trial, the total population size can be significantly reduced while both the same individual and overall success rates are still guaranteed. Experiments showed that in practice step 3 rarely has to be executed, leading to a significant reduction in population size with almost no computational overhead. Furthermore, when step 3 is needed we can efficiently adjust N through a binary search within its bounds, because we know that a larger population size always yields a higher probability to observe all targets.

The joint success probability $Pr[|G_1| \geqslant f_1 \;\&\; \cdots \;\&\; |G_m| \geqslant f_m; N]$ of obtaining each targeted phase-known genotype G_i at least f_i times, when growing N plants in total, is computed as

$$\begin{aligned}
& Pr \quad [|G_1| \geqslant f_1 \;\&\; \cdots \;\&\; |G_m| \geqslant f_m; N] \\
= \;& 1 - \neg Pr[|G_1| \geqslant f_1 \;\&\; \cdots \;\&\; |G_m| \geqslant f_m; N] \\
= \;& 1 - Pr[|G_1| \leqslant (f_1 - 1) \vee \cdots \vee |G_m| \leqslant (f_m - 1); N] \\
= \;& 1 - \sum_{i_1 \in [1,m]} Pr[|G_{i_1}| \leqslant (f_{i_1} - 1); N] \\
+ \;& \sum_{\substack{(i_1,i_2) \in [1,m]^2 \\ i_1 < i_2}} Pr[|G_{i_1}| \leqslant (f_{i_1} - 1) \;\&\; |G_{i_2}| \leqslant (f_{i_2} - 1); N] \\
- \;& \sum_{\substack{(i_1,i_2,i_3) \in [1,m]^3 \\ i_1 < i_2 < i_3}} Pr[|G_{i_1}| \leqslant (f_{i_1} - 1) \;\&\; |G_{i_2}| \leqslant (f_{i_2} - 1) \\
& \qquad\qquad\qquad\qquad\qquad\qquad \;\&\; |G_{i_3}| \leqslant (f_{i_3} - 1); N] \\
+ \;& \cdots \\
& \cdots \\
+ \;& (-1)^m \, Pr[|G_1| \leqslant (f_1 - 1) \;\&\; \cdots \;\&\; |G_m| \leqslant (f_m - 1); N]
\end{aligned}$$

with

$$\begin{aligned}\Pr\ & [|G_{i_1}| \leqslant f_{i_1} \ \&\ \cdots\ \&\ |G_{i_k}| \leqslant f_{i_k}; N] \\ = & \sum_{n_1=0}^{f_{i_1}} \cdots \sum_{n_k=0}^{f_{i_k}} \frac{N \cdot (N-1) \cdots (N-n_1-\cdots-n_k+1)}{n_1! \cdots n_k!} p_{G_{i_1}}^{n_1} \cdots p_{G_{i_k}}^{n_k} \\ & \cdot (1 - p_{G_{i_1}} - \cdots - p_{G_{i_k}})^{N-n_1-\cdots-n_k}\end{aligned}$$

where p_{G_i} is the probability of observing G_i among the offspring. These formulas follow from the multinomial probability distribution. The joint success probability is computed through its complement because the f_i's are expected to be small—often they are all equal to one—while N is expected to be much larger. Also, m is expected to be relatively small. Therefore, a direct computation would require to sum over significantly more terms, as compared to this computation through the complement.

B.3 FULL RESULTS FOR THE FIRST CONSTRUCTED EXAMPLE

Figures B.1 to B.5 show an overview of all five reported solutions for the first constructed example, when running Gene Stacker in *default* mode with an overall success rate of $\gamma = 0.95$ and a maximum of 4 generations, 10% overall linkage phase ambiguity, 4 crossings per plant, 5000 plants per generation, and 2500 seeds obtained from each crossing. Three schemes are non-ambiguous, while the remaining two schemes have a small linkage phase ambiguity of 8.28% which in turn yields a (slightly) lower total population size. The approximated Pareto front clearly reflects the tradeoffs between the three objectives: minimizing the total population size, number of generations, and overall linkage phase ambiguity.

B.4 SPECIFICATION OF REAL STACKING PROBLEMS

This section gives a full description of all discussed problems from cotton, tomato and rice.

Cotton

We used one stacking problem from cotton, consisting of six parental genotypes and a heterozygous ideotype with 11 loci spread across five chromosomes:

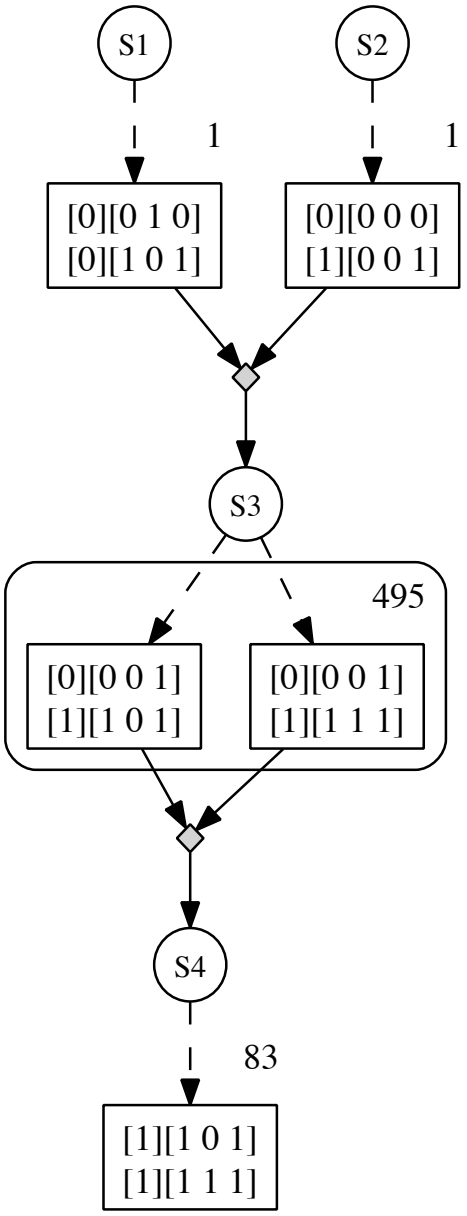
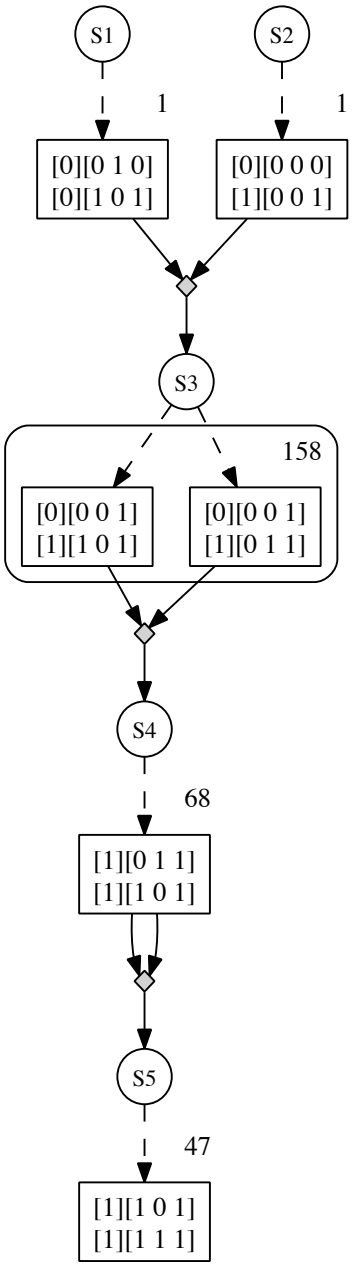
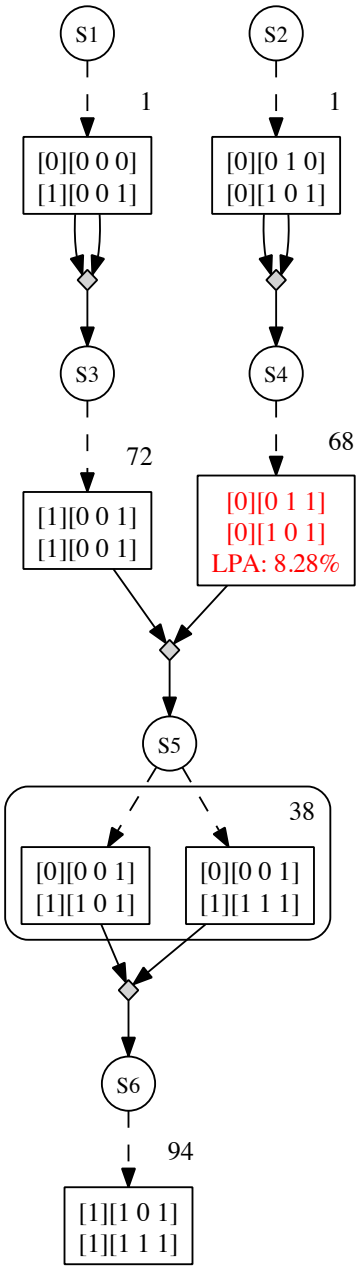


Figure B.1: First reported solution for the first constructed example, when running Gene Stacker in *default* mode with an overall success rate of $\gamma = 0.95$, and a maximum of 4 generations, 10% overall linkage phase ambiguity, 4 crossings per plant, 5000 plants per generation, and 2500 seeds obtained from each crossing.



Overall LPA: 0%
Plants: 275

Figure B.2: Second reported solution for the first constructed example, when running Gene Stacker in *default* mode with an overall success rate of $\gamma = 0.95$, and a maximum of 4 generations, 10% overall linkage phase ambiguity, 4 crossings per plant, 5000 plants per generation, and 2500 seeds obtained from each crossing.



Overall LPA: 8.28%
Plants: 274

Figure B.3: Third reported solution for the first constructed example, when running Gene Stacker in *default* mode with an overall success rate of $\gamma = 0.95$, and a maximum of 4 generations, 10% overall linkage phase ambiguity, 4 crossings per plant, 5000 plants per generation, and 2500 seeds obtained from each crossing.

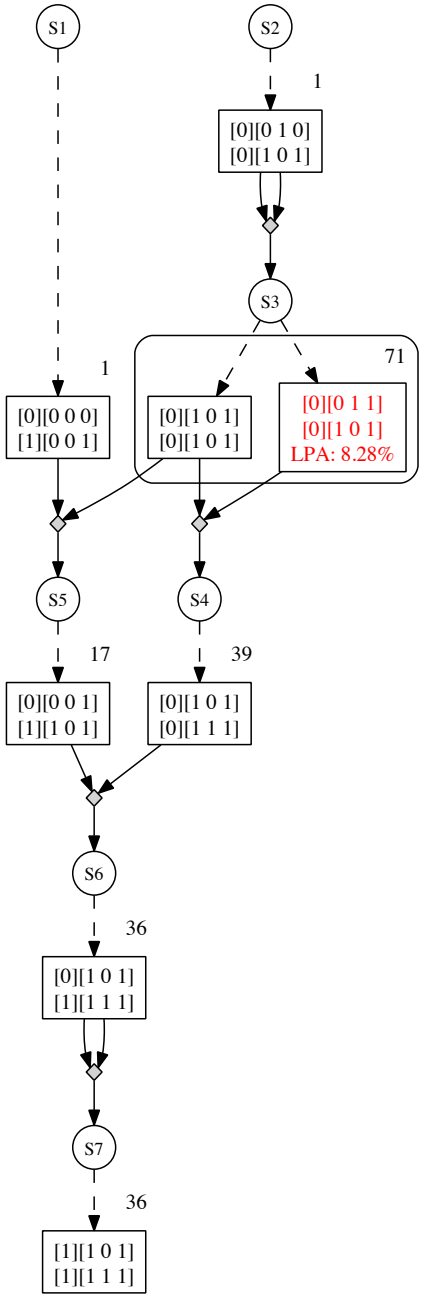


Figure B.4: Fourth reported solution for the first constructed example, when running Gene Stacker in *default* mode with an overall success rate of $\gamma = 0.95$, and a maximum of 4 generations, 10% overall linkage phase ambiguity, 4 crossings per plant, 5000 plants per generation, and 2500 seeds obtained from each crossing.

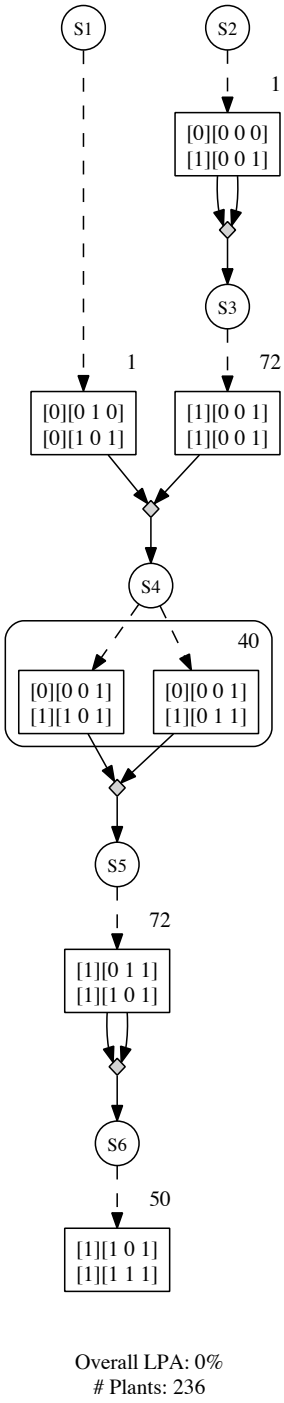


Figure B.5: Fifth reported solution for the first constructed example, when running Gene Stacker in *default* mode with an overall success rate of $\gamma = 0.95$, and a maximum of 4 generations, 10% overall linkage phase ambiguity, 4 crossings per plant, 5000 plants per generation, and 2500 seeds obtained from each crossing.

$$
\begin{aligned}
G^1 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \\
G^2 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
G^3 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
G^4 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
G^5 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \\
G^6 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
\mathcal{J} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}.
\end{aligned}
$$

The genetic map states the following distances between subsequent loci on the same chromosome:

- 2nd chromosome: 15 cM, 10 cM
- 3rd chromosome: 10 cM
- 4th chromosome: 8 cM
- 5th chromosome: 45 cM, 10 cM

Tomato

Both considered stacking problems from tomato consist of the same four parental genotypes with eight loci spread across six chromosomes:

$$
\begin{aligned}
G^1 &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \\
G^2 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},
\end{aligned}
$$

$$G^3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},$$

$$G^4 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

The genetic map states the following distances between subsequent loci on the same chromosome:

- 2nd chromosome: 4 cM
- 6th chromosome: 10 cM

The first problem (Tomato-1) has a homozygous ideotype

$$\mathcal{I} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

while the second problem (Tomato-2) has a heterozygous ideotype

$$\mathcal{I} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Rice

The two considered problems from rice have the same eight parental genotypes with ten loci spread across six chromosomes:

$$G^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

$$G^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$G^3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$G^4 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

$$G^5 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$G^6 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$G^7 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$G^8 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

The genetic map states the following distances between subsequent loci on the same chromosome:

- 3rd chromosome: 5 cM
- 4th chromosome: 9 cM
- 5th chromosome: 50 cM, 8 cM

The first problem (Rice-1) has a homozygous ideotype

$$\mathcal{I} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

while the second problem (Rice-2) has a heterozygous ideotype

$$\mathcal{I} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

B.5 SOLUTIONS FOR REAL STACKING PROBLEM FROM COTTON

Figures B.6 to B.9 show all solutions reported for the cotton example when applying preset *fastest* with a maximum of five generations. Running this preset took 2 hours and 15 minutes to complete; all other presets ran out of memory.

Because only 250 seeds are produced per crossing, some crossings are performed multiple times to provide a sufficient amount of seeds so that all selection targets are expected among the offspring. Furthermore, a cotton plant can only be crossed twice (or selfed once). Therefore, for some genotypes, several duplicates are grown to be able to perform all crossings. Population sizes are computed in such way that at least the required number of occurrences of each selection target is expected among the offspring (see appendix B.2).

When restricting the number of generations to four instead of five, preset *faster* finds a different scheme with four generations, as shown in figure B.10, before being interrupted when the time limit of 24 hours is exceeded. This solution has a lower population size as compared to the respective scheme found by preset *fastest*.

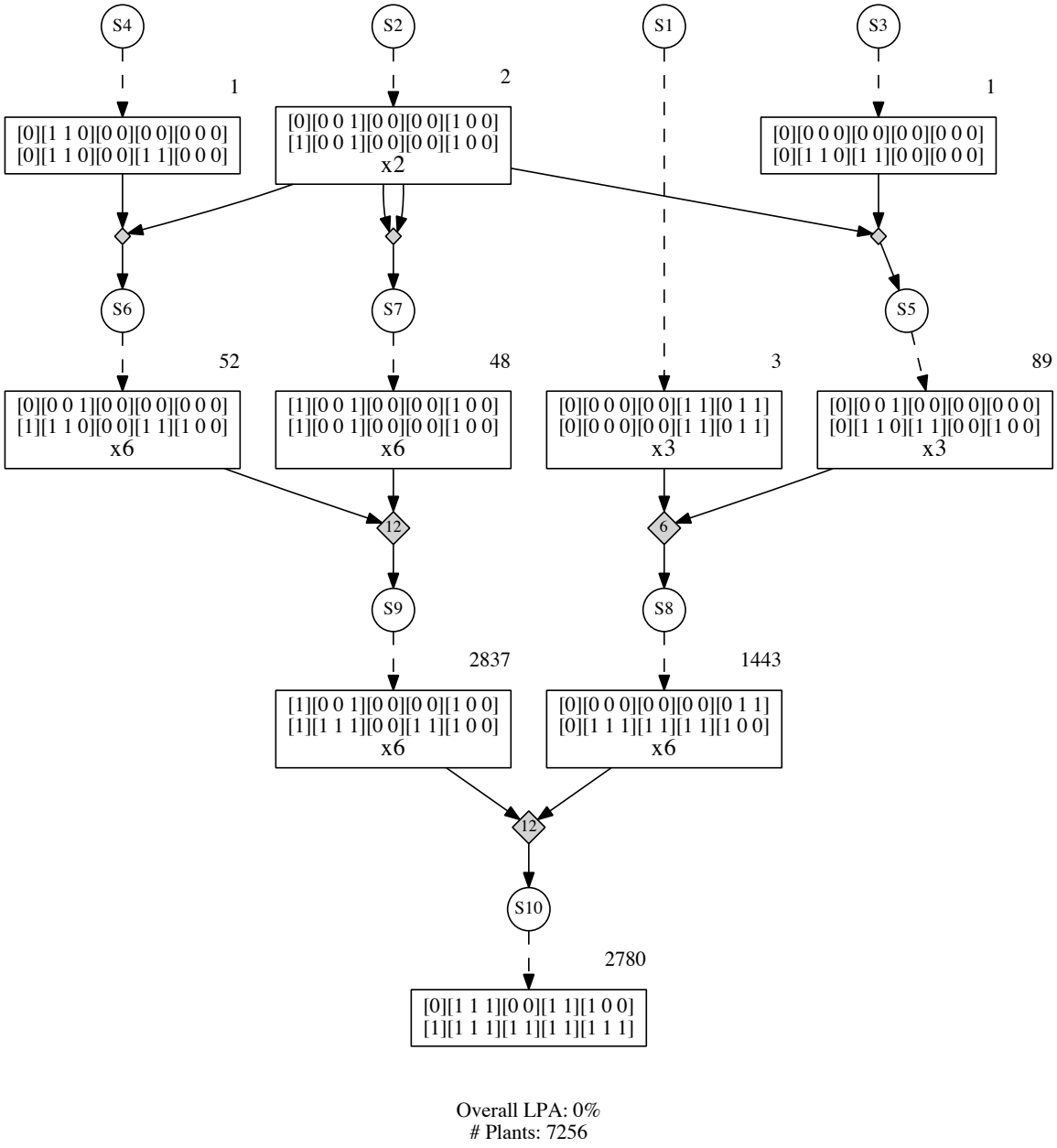


Figure B.6: Three generation long scheme for the cotton example. Some crossings are performed 6 up to 12 times to provide a sufficient amount of seeds. For most genotypes occurring in the scheme, several duplicates are targeted to be able to perform all crossings.

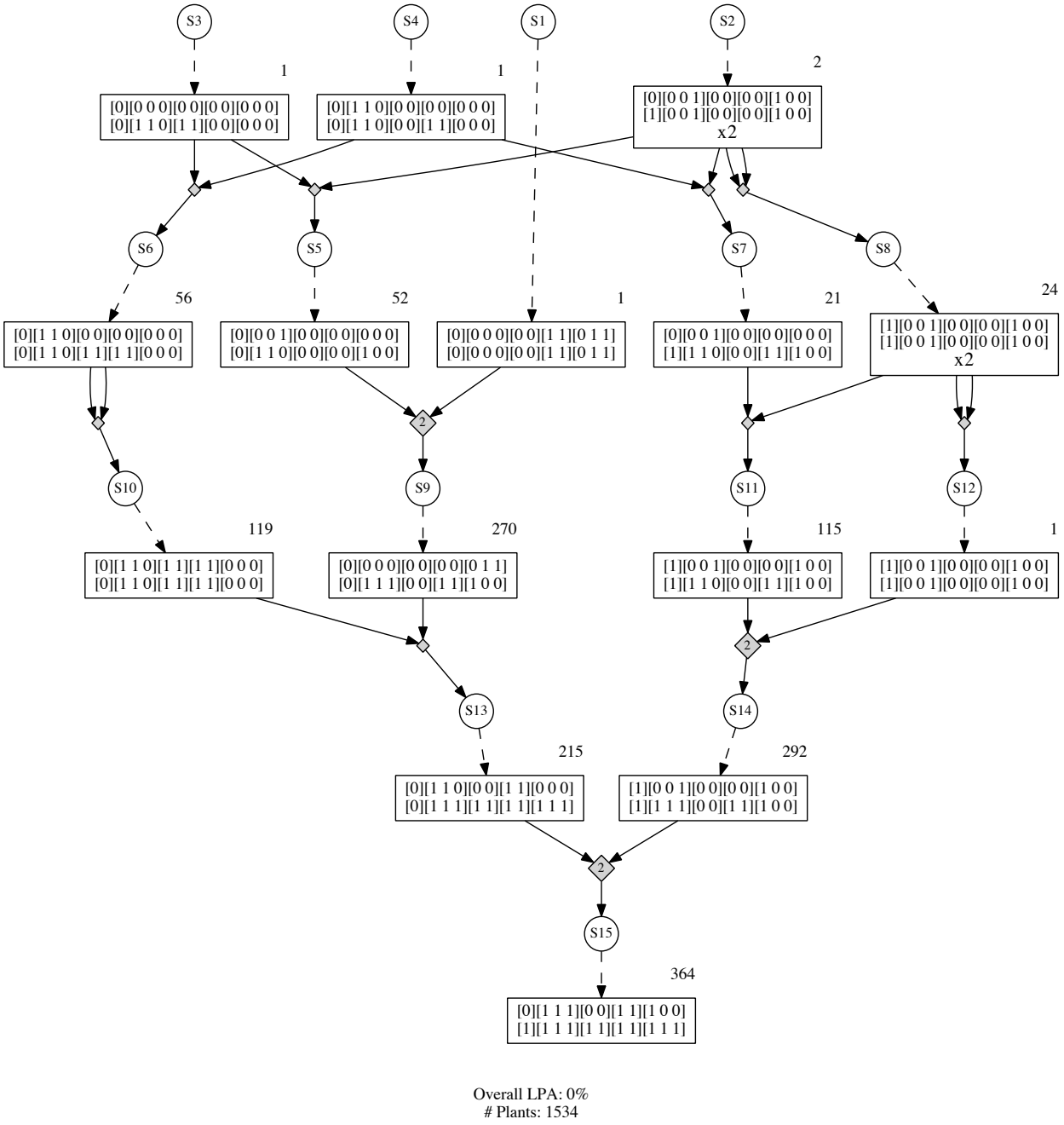


Figure B.7: Four generation long scheme for the cotton example. Some crossings are performed twice to obtain a sufficient amount of seeds. For two genotypes occurring in the scheme, two duplicates are targeted to be able to perform all crossings.

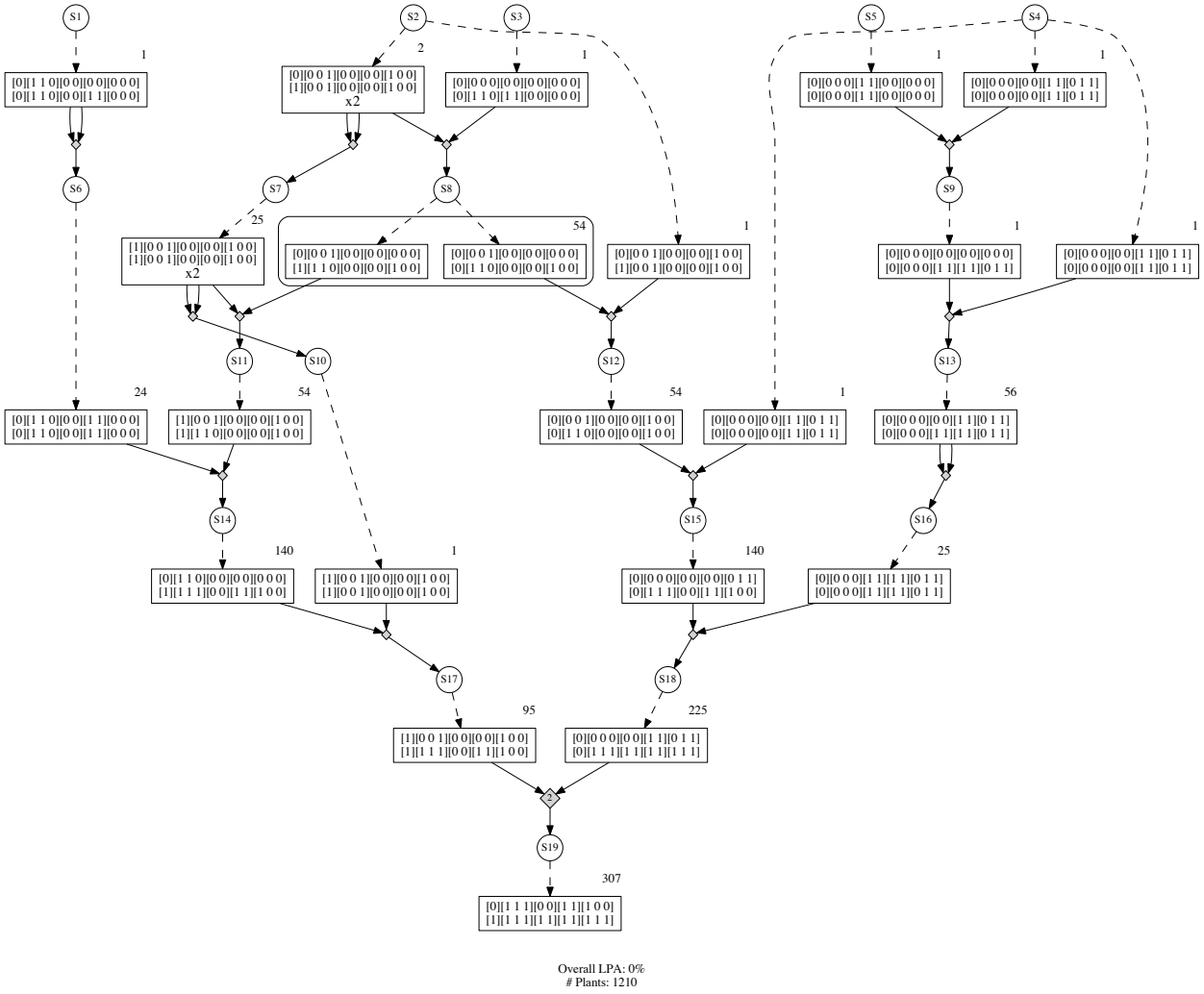


Figure B.8: Five generation long scheme for the cotton example with zero linkage phase ambiguity. Only the final crossing is performed twice to provide a sufficient amount of seeds. For two genotypes occurring in the scheme, two duplicates are targeted to be able to perform all crossings.

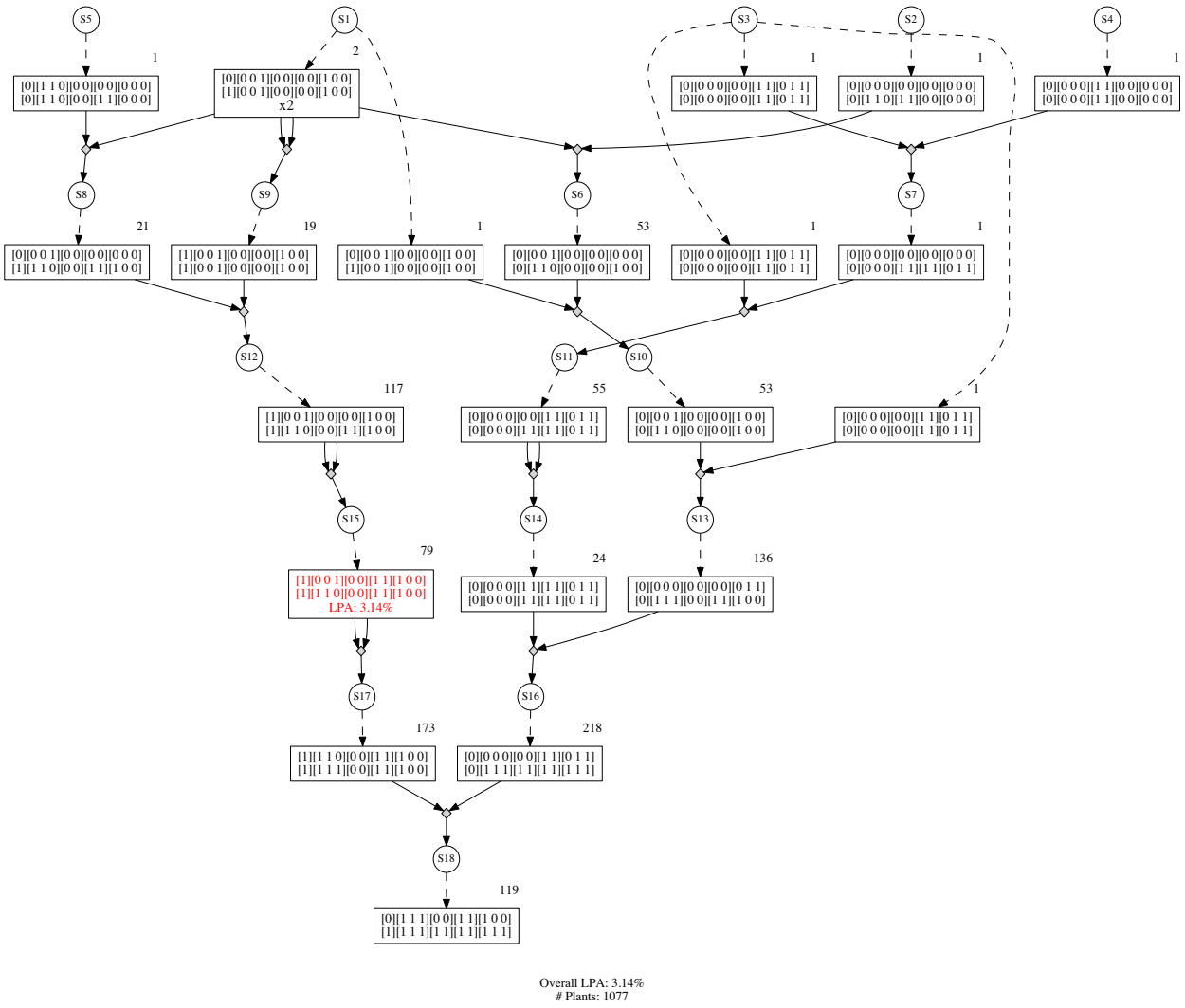


Figure B.9: Five generation long scheme for the cotton example with an overall linkage phase ambiguity of 3.14%. In return, a reduction in the total population size is obtained as compared to the reported non-ambiguous scheme with five generations. No crossings are performed multiple times in this scheme, as a single crossing always provides enough seeds to obtain all targeted genotypes.

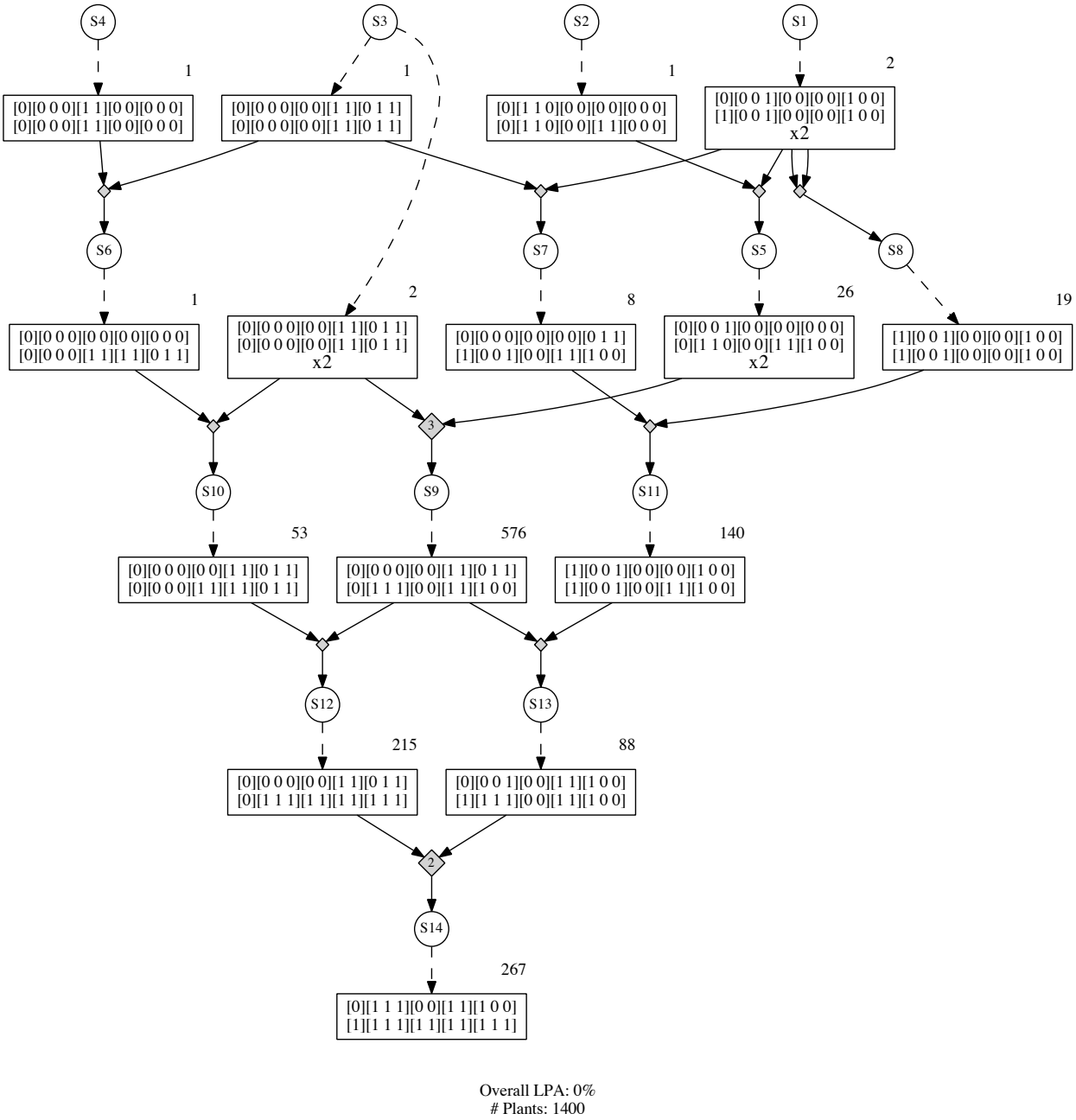


Figure B.10: Additional four generation long scheme for the cotton example, reported by preset *faster* when restricting the number of generations to four instead of five. This scheme has a lower total population size as compared to the solution with four generations that is found by preset *fastest*.

BIBLIOGRAPHY

Acquaah, George (2012). *Principles of plant genetics and breeding*. 2nd ed. John Wiley & Sons.

Ahlinder, J, TJ Mullin and Makoto Yamashita (2014). „Using semidefinite programming to optimize unequal deployment of genotypes to a clonal seed orchard". In: *Tree genetics & genomes* 10.1, pp. 27–34.

Alba, Enrique, Gabriel Luque, Jose Garcia-Nieto and Guillermo Ordonez (2007). „MALLBA: a software library to design efficient optimisation algorithms". In: *International Journal of Innovative Computing and Applications* 1.1, pp. 74–85.

Badke, Yvonne M, Ronald O Bates, Catherine W Ernst, Justin Fix and Juan P Steibel (2014). „Accuracy of estimation of genomic breeding values in pigs using low-density genotypes and imputation". In: *G3: Genes| Genomes| Genetics* 4.4, pp. 623–631.

Berg, Edward E and JL Hamrick (1997). „Quantification of genetic diversity at allozyme loci". In: *Canadian Journal of Forest Research* 27.3, pp. 415–424.

Bertsekas, Dimitri P (2014). *Constrained optimization and Lagrange multiplier methods*. Academic press.

Blake, Victoria C, Jennifer G Kling, Patrick M Hayes, Jean-Luc Jannink, Suman R Jillella, John Lee, David E Matthews, Shiaoman Chao, Timothy J Close, Gary J Muehlbauer et al. (2012). „The hordeum toolbox: the Barley coordinated agricultural project genotype and phenotype resource". In: *The Plant Genome* 5.2, pp. 81–91.

Borrayo, Ernesto, Ryoko Machida-Hirano, Masaru Takeya, Makoto Kawase and Kazuo Watanabe (2016). „Principal components analysis-K-means transposon element based foxtail millet core collection selection method". In: *BMC genetics* 17.1, p. 1.

Brown, AHD (1989). „Core collections: a practical approach to genetic resources management". In: *Genome* 31.2, pp. 818–824.

Browning, Brian L and Sharon R Browning (2009). „A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals". In: *The American Journal of Human Genetics* 84.2, pp. 210–223.

Browning, Sharon R and Brian L Browning (2007). „Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering". In: *The American Journal of Human Genetics* 81.5, pp. 1084–1097.

Browning, Sharon R and Brian L Browning (2011). „Haplotype phasing: existing methods and new developments". In: *Nature Reviews Genetics* 12.10, pp. 703–714.

Brownlee, Jason (2007). „Oat: The optimization algorithm toolkit". In: *Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report A* 20071220.

Cahon, Sébastien, Nordine Melab and E-G Talbi (2004). „Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics". In: *Journal of Heuristics* 10.3, pp. 357–380.

Camerarius, Rudolph J (1694). „De sexu plantarum epistola". In: *Ostwald's Klassiker der exakten Naturwissenschaften (reissued: 1899)*.

Campos, Gustavo de los and Paulino Pérez (2015). *BGLR: Bayesian Generalized Linear Regression*. R package version 1.0.4. URL: http://CRAN.R-project.org/package=BGLR.

Cantor, Rita M, Kenneth Lange and Janet S Sinsheimer (2010). „Prioritizing GWAS results: a review of statistical methods and recommendations for their application". In: *The American Journal of Human Genetics* 86.1, pp. 6–22.

Canzar, Stefan and Mohammed El-Kebir (2011). „A mathematical programming approach to marker-assisted gene pyramiding". In: *Algorithms in Bioinformatics, WABI 2011, LNBI 6833*. Ed. by Teresa M Przytycka and Marie-France Sagot. Heidelberger Platz 3, 14197 Berlin, Germany: Springer, pp. 26–38.

Carey, James and Brent Carlson (2002). „Lessons learned becoming a framework developer". In: *Software: Practice and Experience* 32.8, pp. 789–800.

Carvalheiro, Roberto, Sandra Aidar de Queiroz and Brian Kinghorn (2010). „Optimum contribution selection using differential evolution". In: *Revista Brasileira de Zootecnia* 39.7, pp. 1429–1436.

Cavalli-Sforza, Luigi L and Anthony WF Edwards (1967). „Phylogenetic analysis. Models and estimation procedures". In: *American journal of human genetics* 19.3 Pt 1, p. 233.

Clark, Samuel A, Brian P Kinghorn, John M Hickey and Julius HJ van der Werf (2013). „The effect of genomic information on optimal contribution selection in livestock breeding programs". In: *Genetics Selection Evolution* 45.1, p. 1.

Close, Timothy J, Prasanna R Bhat, Stefano Lonardi, Yonghui Wu, Nils Rostoks, Luke Ramsay, Arnis Druka, Nils Stein, Jan T Svensson, Steve Wanamaker et al. (2009). „Development and implementation of high-throughput SNP genotyping in barley". In: *BMC genomics* 10.1, p. 582.

Daetwyler, Hans D, Mario PL Calus, Ricardo Pong-Wong, Gustavo de los Campos and John M Hickey (2013). „Genomic prediction in animals and plants: simulation of data, validation, reporting, and benchmarking". In: *Genetics* 193.2, pp. 347–365.

Darwin, Charles (1859). „On the origin of species: a facsimile of the first edition with an introduction by Ernst Mayr". In: *New York: Atheneum (1967)*.

Das, Indraneel and John E Dennis (1998). „Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems". In: *SIAM Journal on Optimization* 8.3, pp. 631–657.

Dawkins, Paul (2016). *Paul's Online Math Notes: Calculus I – Optimization*. URL: http://tutorial.math.lamar.edu/Classes/CalcI/Optimization.aspx (visited on 02/12/2016).

De Beukelaer, Herman, Petr Smỳkal, Guy F Davenport and Veerle Fack (2012). „Core Hunter II: fast core subset selection based on multiple genetic diversity measures using Mixed Replica search". In: *BMC bioinformatics* 13.1, p. 312.

De Beukelaer, Herman, Guy F Davenport, Geert De Meyer and Veerle Fack (2017). „JAMES: An object-oriented Java framework for discrete optimization using local search metaheuristics". In:

Software: Practice and Experience 47.6, pp. 921–938. DOI: 10.1002/spe.2459.

De Corte, Annelies and Kenneth Sörensen (2016). „An iterated local search algorithm for water distribution network design optimization". In: *Networks* 67.3, pp. 187–198.

Deb, Kalyanmoy, Amrit Pratap, Sameer Agarwal and TAMT Meyarivan (2002). „A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2, pp. 182–197.

Di Gaspero, Luca and Andrea Schaerf (2003). „EASYLOCAL++: an object-oriented framework for the flexible design of local-search algorithms". In: *Software: Practice and Experience* 33.8, pp. 733–765.

Doudna, Jennifer A and Emmanuelle Charpentier (2014). „The new frontier of genome engineering with CRISPR-Cas9". In: *Science* 346.6213, p. 1258096.

Durillo, Juan J and Antonio J Nebro (2011). „jMetal: A Java framework for multi-objective optimization". In: *Advances in Engineering Software* 42.10, pp. 760–771.

Earl, David J and Michael W Deem (2005). „Parallel tempering: Theory, applications, and new perspectives". In: *Physical Chemistry Chemical Physics* 7.23, pp. 3910–3916.

El-Kebir, Mohammed, MT de Berg and JB Buntjer (2009). „Crossing Schedule Optimization". MA thesis. Technische Universiteit Eindhoven.

Endelman, JB (2011). „Ridge regression and other kernels for genomic selection with R package rrBLUP". In: *Plant Genome* 4, pp. 250–255.

Erfani, Tohid and Sergei V Utyuzhnikov (2011). „Directed search domain: a method for even generation of the Pareto frontier in multiobjective optimization". In: *Engineering Optimization* 43.5, pp. 467–484.

Falconer, Douglas S, Trudy FC Mackay and Richard Frankham (1996). „Introduction to quantitative genetics (4th edn)". In: *Trends in Genetics* 12.7, p. 280.

Fisher, Ronald A (1919). „XV. The correlation between relatives on the supposition of Mendelian inheritance." In: *Transactions of the royal society of Edinburgh* 52.02, pp. 399–433.

Fortnow, Lance (2009). „The status of the P versus NP problem". In: *Communications of the ACM* 52.9, pp. 78–86.

Franco, Jorge, José Crossa, Suketoshi Taba and Henry Shands (2005). „A sampling strategy for conserving genetic diversity when forming core subsets". In: *Crop Science* 45.3, pp. 1035–1044.

Franco, Jorge, José Crossa and Santosh Desphande (2010). „Hierarchical multiple-factor analysis for classifying genotypes based on phenotypic and genetic data". In: *Crop Science* 50.1, pp. 105–117.

Frankel, OH et al. (1984). „Genetic perspectives of germplasm conservation". In: *Genetic manipulation: impact on man and society. Cambridge University Press, Cambridge*, pp. 161–170.

Gendreau, Michel and Jean-Yves Potvin (2010). *Handbook of metaheuristics*. Vol. 2. Springer.

Gilbert, Paul (2014). *setRNG: Set (Normal) Random Number Generator and Seed*. R package version 2013.9-1. URL: http://CRAN.R-project.org/package=setRNG.

Glover, Fred and Eric Taillard (1993). „A user's guide to tabu search". In: *Annals of operations research* 41.1, pp. 1–28.

Goddard, Mike (2009). „Genomic selection: prediction of accuracy and maximisation of long term response". In: *Genetica* 136.2, pp. 245–257.

Gouesnard, B, TM Bataillon, G Decoux, C Rozale, DJ Schoen and JL David (2001). „MSTRAT: An algorithm for building germ plasm core collections by maximizing allelic or phenotypic richness". In: *Journal of Heredity* 92.1, pp. 93–94.

Gower, John C (1971). „A general coefficient of similarity and some of its properties". In: *Biometrics*, pp. 857–871.

Grundy, B, B Villanueva and JA Woolliams (1998). „Dynamic selection procedures for constrained inbreeding and their consequences for pedigree development". In: *Genetical Research* 72.02, pp. 159–168.

Haldane, JBS (1919). „The combination of linkage values and the calculation of distances between the loci of linked factors". In: *J Genet* 8.29, pp. 299–309.

Hansen, Pierre, Nenad Mladenović and José A Moreno Pérez (2010). „Variable neighbourhood search: methods and applications". In: *Annals of Operations Research* 175.1, pp. 367–407.

Harrell, Frank E Jr, with contributions from Charles Dupont and many others. (2015). *Hmisc: Harrell Miscellaneous*. R package version 3.16-0. URL: http://CRAN.R-project.org/package=Hmisc.

Hayes, BJ, PJ Bowman, AJ Chamberlain and ME Goddard (2009). „Invited review: Genomic selection in dairy cattle: Progress and challenges". In: *Journal of dairy science* 92.2, pp. 433–443.

Heffner, Elliot L, Mark E Sorrells and Jean-Luc Jannink (2009). „Genomic selection for crop improvement". In: *Crop Science* 49.1, pp. 1–12.

Holland, John H (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Hsu, Patrick D, Eric S Lander and Feng Zhang (2014). „Development and applications of CRISPR-Cas9 for genome engineering". In: *Cell* 157.6, pp. 1262–1278.

Ishii, T and K Yonezawa (2007). „Optimization of the marker-based procedures for pyramiding genes from multiple donor lines: I. Schedule of crossing between the donor lines". In: *Crop science* 47.2, pp. 537–546.

Jannink, Jean-Luc (2010). „Dynamics of long-term genomic selection". In: *Genetics Selection Evolution* 42.1, p. 35.

Jansen, J and TH Van Hintum (2007). „Genetic distance sampling: a novel sampling method for obtaining core collections using genetic distances with an application to cultivated lettuce". In: *Theoretical and Applied Genetics* 114.3, pp. 421–428.

Jing, Runchun, Alexander Vershinin, Jacek Grzebyta, Paul Shaw, Petr Smỳkal, David Marshall, Michael J Ambrose, TH Noel Ellis and Andrew J Flavell (2010). „The genetic diversity and

evolution of field pea (Pisum) studied by high throughput retrotransposon based insertion polymorphism (RBIP) marker analysis". In: *BMC Evolutionary Biology* 10.1, p. 1.

Johnson, David S and Lyle A McGeoch (1997). „The traveling salesman problem: A case study in local optimization". In: *Local search in combinatorial optimization* 1, pp. 215–310.

Kaufman, Leonard and Peter J Rousseeuw (1990). „Partitioning around medoids (program pam)". In: *Finding groups in data: an introduction to cluster analysis*, pp. 68–125.

Kennedy, James (2011). „Particle swarm optimization". In: *Encyclopedia of machine learning*. Springer, pp. 760–766.

Kim, Kyu-Won, Hun-Ki Chung, Gyu-Taek Cho, Kyung-Ho Ma, Dorothy Chandrabalan, Jae-Gyun Gwag, Tae-San Kim, Eun-Gi Cho and Yong-Jin Park (2007). „PowerCore: a program applying the advanced M strategy with a heuristic search for establishing core sets". In: *Bioinformatics* 23.16, pp. 2155–2162.

Kinghorn, Brian P (2011). „An algorithm for efficient constrained mate selection". In: *Genetics Selection Evolution* 43.1, p. 1.

Kirkpatrick, Scott, C Daniel Gelatt and Mario P Vecchi (1983). „Optimization by simulated annealing". In: *science* 220.4598, pp. 671–680.

Krishnan, Ramesh R, R Sumathy, SR Ramesh, BB Bindroo and Girish V Naik (2014). „SimEli: Similarity Elimination method for sampling distant entries in development of core collections". In: *Crop Science* 54.3, pp. 1070–1078.

Kronfeld, Marcel, Hannes Planatscher and Andreas Zell (2010). „The EvA2 optimization framework". In: *Learning and Intelligent Optimization*. Springer, pp. 247–250.

Li, Y, HN Kadarmideen and JCM Dekkers (2008). „Selection on multiple QTL with control of gene diversity and inbreeding for long-term benefit". In: *Journal of Animal Breeding and Genetics* 125.5, pp. 320–329.

Lindgren, D and TJ Mullin (1997). „Balancing gain and relatedness in selection". In: *Silvae Genetica* 46.2, pp. 124–128.

Liu, AYH and JA Woolliams (2010). „Continuous approximations for optimizing allele trajectories". In: *Genetics research* 92.02, pp. 157–166.

Liu, Huiming, Theo Meuwissen, Anders C Sørensen and Peer Berg (2015). „Upweighting rare favourable alleles increases long-term genetic gain in genomic selection programs". In: *Genetics Selection Evolution* 47.1, p. 19.

Lukasiewycz, Martin, Michael Glaß, Felix Reimann and Jürgen Teich (2011). „Opt4J: a modular framework for meta-heuristic optimization". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, pp. 1723–1730.

Marler, R Timothy and Jasbir S Arora (2005). „Function-transformation methods for multi-objective optimization". In: *Engineering Optimization* 37.6, pp. 551–570.

Marler, R Timothy and Jasbir S Arora (2010). „The weighted sum method for multi-objective optimization: new insights". In: *Structural and multidisciplinary optimization* 41.6, pp. 853–862.

Messac, Achille and Christopher A Mattson (2004). „Normal constraint method with guarantee of even representation of complete Pareto frontier". In: *AIAA journal* 42.10, pp. 2101–2111.

Meuwissen, Theo (1997). „Maximizing the response of selection with a predefined rate of inbreeding." In: *Journal of animal science* 75.4, pp. 934–940.

Meuwissen, Theo (2002). „GENCONT: an operational tool for controlling inbreeding in selection and conservation schemes". In: *Proceedings of 7th World Congr Genet Appl Livest Prod, Montpellier*, pp. 769–770.

Meuwissen, Theo, Ben J Hayes and Mike E Goddard (2001). „Prediction of total genetic value using genome-wide dense marker maps". In: *Genetics* 157.4, pp. 1819–1829.

Motta, Renato de S, Silvana MB Afonso and Paulo RM Lyra (2012). „A modified NBI and NC method for the solution of N-multiobjective optimization problems". In: *Structural and Multidisciplinary Optimization* 46.2, pp. 239–259.

Mueller-Gritschneder, Daniel, Helmut Graeb and Ulf Schlichtmann (2009). „A successive approach to compute the bounded Pareto front of practical multiobjective optimization problems". In: *SIAM Journal on Optimization* 20.2, pp. 915–934.

Mullin, TJ and P Belotti (2016). „Using branch-and-bound algorithms to optimize selection of a fixed-size breeding population under a relatedness constraint". In: *Tree Genetics & Genomes* 12.1, pp. 1–12.

Odong, TL, J van Heerwaarden, J Jansen, Th JL van Hintum and FA van Eeuwijk (2011). „Statistical techniques for defining reference sets of accessions and microsatellite markers". In: *Crop science* 51.6, pp. 2401–2411.

Odong, TL, J Jansen, FA Van Eeuwijk and Th JL van Hintum (2013). „Quality of core collections for effective utilisation of genetic resources review, discussion and interpretation". In: *Theoretical and Applied Genetics* 126.2, pp. 289–305.

Parejo, José Antonio, Jesús Racero, Fernando Guerrero, Terence Kwok and Kate A Smith (2003). „Fom: A framework for metaheuristic optimization". In: *Computational Science – ICCS 2003*. Springer, pp. 886–895.

Parejo, José Antonio, Antonio Ruiz-Cortés, Sebastián Lozano and Pablo Fernandez (2012). „Metaheuristic optimization frameworks: a survey and benchmarking". In: *Soft Computing* 16.3, pp. 527–561.

Patriksson, M, N Andreasson and A Evgrafov (2017). *An Introduction to Continuous Optimization: Foundations and Fundamental Algorithms, Third Edition*. Dover Books on Mathematics Series. Dover Publications. ISBN: 9780486802879. URL: https://books.google.be/books?id=C9JgrgEACAAJ.

Plummer, Martyn, Nicky Best, Kate Cowles and Karen Vines (2006). „CODA: Convergence Diagnosis and Output Analysis for MCMC". In: *R News* 6.1, pp. 7–11. URL: http://CRAN.R-project.org/doc/Rnews/.

Poland, Jesse A and Trevor W Rife (2012). „Genotyping-by-sequencing for plant breeding and genetics". In: *The Plant Genome* 5.3, pp. 92–102.

Pong-Wong, Ricardo and John A Woolliams (2007). „Optimisation of contribution of candidate parents to maximise genetic gain and restricting inbreeding using semidefinite programming (Open Access publication)". In: *Genetics Selection Evolution* 39.1, p. 1.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: http://www.R-project.org/ (visited on 12/10/2016).

Reinelt, Gerhard (1991). „TSPLIB - A traveling salesman problem library". In: *ORSA journal on computing* 3.4, pp. 376–384.

Schoen, Daniel J and AH Brown (1993). „Conservation of allelic richness in wild crop relatives is aided by assessment of genetic markers". In: *Proceedings of the national academy of sciences* 90.22, pp. 10623–10627.

Semagn, Kassa, Å Bjørnstad and MN Ndjiondjop (2006). „An overview of molecular marker methods for plants". In: *African journal of biotechnology* 5.25.

Servin, Bertrand, Olivier C Martin, Marc Mézard and Frédéric Hospital (2004). „Toward a Theory of Marker-Assisted Gene Pyramiding". In: *Genetics* 168, pp. 513–523.

Shannon, Claude Elwood (2001). „A mathematical theory of communication". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1, pp. 3–55.

Smỳkal, Petr, Miroslav Hỳbl, Jukka Corander, Jiří Jarkovskỳ, Andrew J Flavell and Miroslav Griga (2008). „Genetic diversity and population structure of pea (Pisum sativum L.) varieties derived from combined retrotransposon, microsatellite and morphological marker analysis". In: *Theoretical and Applied Genetics* 117.3, pp. 413–424.

Smỳkal, Petr, Gregory Kenicer, Andrew J Flavell, Jukka Corander, Oleg Kosterin, Robert J Redden, Rebecca Ford, Clarice J Coyne, Nigel Maxted, Mike J Ambrose et al. (2011). „Phylogeny, phylogeography and genetic diversity of the Pisum genus". In: *Plant Genetic Resources* 9.01, pp. 4–18.

Sonesson, Anna K, John A Woolliams and Theo Meuwissen (2012). „Genomic selection requires genomic control of inbreeding". In: *Genetics Selection Evolution* 44.1, pp. 1–10.

Storn, Rainer and Kenneth Price (1997). „Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11.4, pp. 341–359.

Sun, Chuanyu and Paul M VanRaden (2014). „Increasing Long-Term Response by Selecting for Favorable Minor Alleles". In: *PLoS ONE* 9.2, e88510. DOI: 10.1371/journal.pone.0088510. URL: http://dx.doi.org/10.1371%2Fjournal.pone.0088510.

Technow, Frank (2014). *hypred: Simulation of Genomic Data in Applied Genetics*. R package version 0.5.

Tester, Mark and Peter Langridge (2010). „Breeding technologies to increase crop production in a changing world". In: *Science* 327.5967, pp. 818–822.

Thachuk, Chris, José Crossa, Jorge Franco, Susanne Dreisigacker, Marilyn Warburton and Guy F Davenport (2009). „Core Hunter: an algorithm for sampling genetic resources based on multiple genetic measures". In: *BMC bioinformatics* 10.1, p. 243.

Urbanek, Simon (2015). *rJava: Low-Level R to Java Interface*. R package version 0.9-7. URL: http://CRAN.R-project.org/package=rJava.

Uzogara, Stella G (2000). „The impact of genetic modification of human foods in the 21st century: A review". In: *Biotechnology advances* 18.3, pp. 179–206.

VanRaden, PM (2008). „Efficient methods to compute genomic predictions". In: *Journal of dairy science* 91.11, pp. 4414–4423.

VanRaden, PM, CP Van Tassell, GR Wiggans, TS Sonstegard, RD Schnabel, JF Taylor and FS Schenkel (2009). „Invited review: Reliability of genomic predictions for North American Holstein bulls". In: *Journal of Dairy Science* 92.1, pp. 16–24.

Ventura, Sebastián, Cristóbal Romero, Amelia Zafra, José A Delgado and César Hervás (2008). „JCLEC: a Java framework for evolutionary computation". In: *Soft Computing* 12.4, pp. 381–392.

Wang, JC, Jin Hu, HM Xu and S Zhang (2007). „A strategy on constructing core collections by least distance stepwise sampling". In: *Theoretical and Applied Genetics* 115.1, pp. 1–8.

Wang, Jian-Cheng, Jin Hu, Ning-Ning Liu, Hai-Ming Xu and Sheng Zhang (2006). „Investigation of combining plant genotypic values and molecular marker information for constructing core subsets". In: *Journal of Integrative Plant Biology* 48.11, pp. 1371–1378.

Warnes, Gregory R, Ben Bolker, Gregor Gorjanc, Gabor Grothendieck, Ales Korosec, Thomas Lumley, Don MacQueen, Arni Magnusson, Jim Rogers and others (2015). *gdata: Various R Programming Tools for Data Manipulation*. R package version 2.17.0. URL: http://CRAN.R-project.org/package=gdata.

Watson, JD and FHC Crick (1953). „Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid". In: *Nature* 171.4356, pp. 737–738. URL: http://dx.doi.org/10.1038/171737a0.

White, David R (2012). „Software review: the ECJ toolkit". In: *Genetic Programming and Evolvable Machines* 13.1, pp. 65–67.

Wientjes, Yvonne CJ, Roel F Veerkamp and Mario PL Calus (2013). „The effect of linkage disequilibrium and family relationships on the reliability of genomic prediction". In: *Genetics* 193.2, pp. 621–631.

Wiggans, GR, PM VanRaden and TA Cooper (2011). „The genomic evaluation system in the United States: Past, present, future". In: *Journal of dairy science* 94.6, pp. 3202–3211.

Wimmer, Valentin, Theresa Albrecht, Hans-Juergen Auinger and Chris-Carolin Schoen (2015). *synbreedData: Data for the Synbreed Package*. R package version 1.5. URL: https://CRAN.R-project.org/package=synbreedData.

Woolliams, JA, P Berg, BS Dagnachew and T Meuwissen (2015). „Genetic contributions and their optimization". In: *Journal of Animal Breeding and Genetics* 132.2, pp. 89–99.

Wright, Sewall (1987). „Evolution and the genetics of population". In: *The theory of gene frequencies* 2.

Xu, Pan, Lizhi Wang and William D Beavis (2011). „An optimization approach to gene stacking". In: *European Journal of Operational Research* 214, pp. 168–178.

Ye, Guoyou and Kevin F Smith (2008). „Marker-assisted gene pyramiding for inbred line development: Basic principles and practical guidelines". In: *International Journal of plant breeding* 2.1, pp. 1–10.

Zitzler, Eckart, Marco Laumanns and Lothar Thiele (2001). „SPEA2: Improving the strength Pareto evolutionary algorithm". In: *Eurogen*. Vol. 3242. 103, pp. 95–100.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available from:

`https://bitbucket.org/amiede/classicthesis/`

Final Version as of 14th June 2017 (`classicthesis` version 4.0).