Crossing schedule optimization

El-Kebir, M.

Award date:
2009

technische universiteit eindhoven
Department of Mathematics and Computer Science

Master's Thesis

Crossing Schedule Optimization

by
Mohammed El-Kebir

Supervisors
prof. dr. M.T. de Berg
dr. J.B. Buntjer

Referee
dr. M.A. Westenberg

Eindhoven, July 30, 2009

This is a public version of the thesis in which confidential information has been omitted.

Abstract

A *genotype* consists of two bitstrings of size m. We are given a set of n genotypes and a desired genotype called the *ideotype*. By crossing two genotypes, new genotypes can be obtained. The probability of obtaining a particular genotype out of two genotypes can be computed in advance. This probability can be translated to a *population size* in which, with a prescribed probability of success, the required genotype is present.

In a *crossing schedule* it is described which crossings are needed in order to obtain the ideotype. Every crossing schedule is assigned a cost; one parameter of the cost function is the total population size needed. We are interested in obtaining the minimum-cost crossing schedule.

In this thesis, the problem is defined formally and an NP-hardness result is given. Two existing methods are described and improved upon. Also two new methods are presented. Subsequently these methods are subjected to an experimental evaluation. The thesis concludes by giving directions for future research.

Acknowledgments

I would like to thank my supervisors Mark de Berg and Jaap Buntjer for their excellent guidance. I am also grateful to Kamyar Malakpoor for his valuable advice and always being willing to answer my questions. The two sessions I had with Rob Bisseling were really helpful and I would like to thank him for that. Finally, I wish to thank José Guerra for teaching me how to build mixed models in SAS and introducing me to Bayesian statistics.

Contents

Chapter 1

Introduction

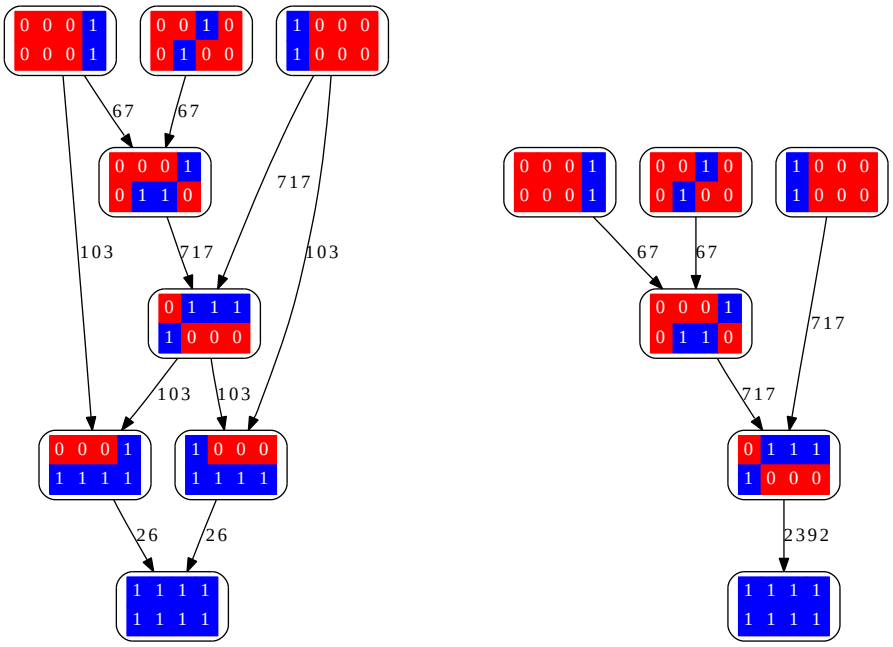
Ever since mankind moved from hunting-gathering to farming, plant breeding has been common practice and resulted in the domestication of various crops as we know them today. At first plant breeding consisted of simply selecting plants that had desirable traits. Later, more sophisticated techniques were used: some Native Americans maintained pure line maize cultivars and used these to generate the heterozygous seeds used on their fields [3]. With the beginning of the plant biotechnology era in the early 1980s, genetic maps containing the locations of markers associated with desirable traits started being developed [13]. For the first time selection at the molecular level was made possible.

Knowledge about which alleles are responsible for certain desired traits allows us to plan, in a systematic way, the process of obtaining an individual in which all the favorable trait alleles are present. Selection now becomes a screening method for finding the intermediary individuals, required by the plan, among the progeny resulting from the performed crossings. The plan can be viewed as a *crossing schedule* that results in an individual with the desired genotype. The desired genotype is called the *ideotype*. A crossing schedule is a description of which crossings are required for obtaining the ideotype. For every crossing it is described which individuals participate in that crossing, what the resulting genotype is and how many progeny is required for obtaining that genotype.

In this thesis we introduce the problem that takes as input the genetic map, an initial set of parents with known genotypes and the ideotype. Using the former, we have to come up with a crossing schedule that results in the ideotype. In Figure 1.1 two example crossing schedules are shown. These two crossing schedules involve the same set of original parents and result in the same ideotype, yet their topologies differ. Indeed, there are many possible crossing schedules, each having its own characteristics. The characteristics that we are interested in and want to optimize are as follows.

- Number of generations
  It takes time for the progeny to mature such that a next crossing can be performed. For instance, one generation takes 2-3 months for Arabidopsis, while for trees it takes 10-20 years. So the number of generations needed for a crossing schedule is a measure on the time it takes to actually perform it.

- Number of crossings
  Every specific crossing between two individuals requires an effort from the breeder, e.g.



(a) Without selfing, requires 4 generations, 5 crossings and a total population size of 816.

(b) With selfing, requires 3 generations, 3 crossings and a total population size of 3176.

Figure 1.1: Example crossing schedules. The genotypes of the three parents are in the first generation, whereas the final generation contains the ideotype.

the plants have to made to flower at the same time. So typically we are also interested in obtaining a crossing schedule with as few crossings as possible.

- Total population size
  In order to obtain the desired genotype from a crossing, a specific number of offspring needs to be generated among which the desired genotype is expected to be present. The more difficult a genotype is to obtain out of its parents, the larger the number of required offspring will be. Every individual in the offspring has to be grown and screened for having the desired genotype. So the more individuals a crossing schedule requires, the more expensive it will be to perform in practice.

  Plants only allow a limited number of offspring; the actual number varies between species. We can take this into account by only allowing crossing steps with a bounded maximum population size.

Let's take a closer look at the two crossing schedules in Figure 1.1. A directed edge relates an individual with its direct descendant; the label indicates the population size needed for obtaining the descendant from its parents. An individual can have a single parent instead of two. This is because some plants allow for self-pollination (called *selfing*). The ideotype in the right-hand schedule is obtained via a selfing step as the last step in the schedule. The right-hand schedule requires one generation and two crossings less than the left-hand schedule. However, the total population size needed is four times bigger than the left-hand one. This is largely due to the final selfing step that requires a population size of 2392, which

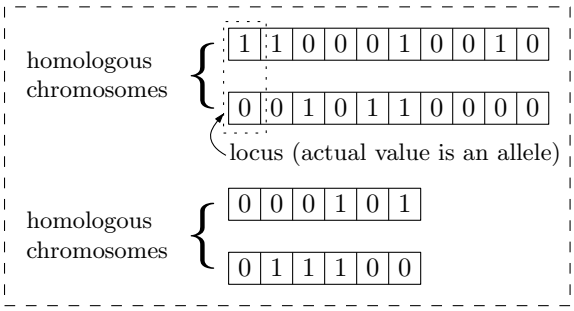


Figure 1.2: Example of a diploid cell with two pairs of homologous chromosomes containing ten and six loci respectively. A chromosome is represented as a string over some alphabet, a locus is an index and an allele is the actual value at some index. Note that a locus does not necessarily correspond to a gene (it can also correspond to a marker).

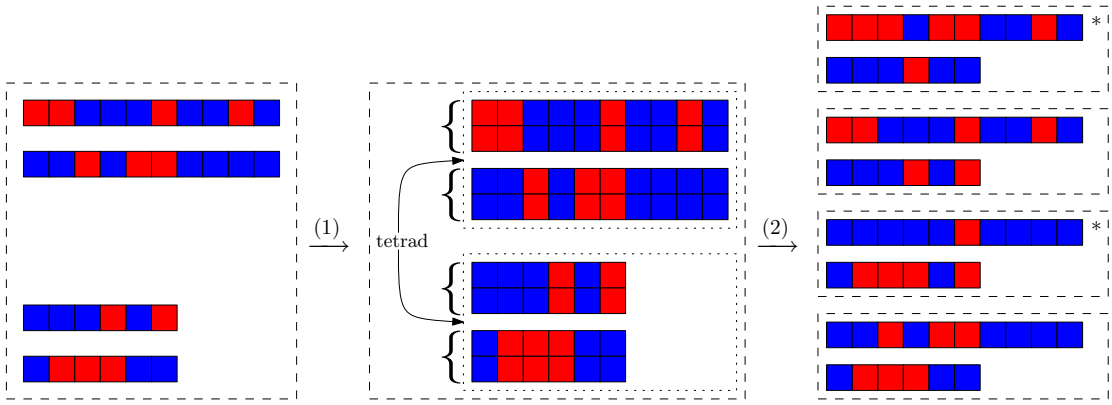
may be infeasible in practice. Depending on what characteristics are more important one might prefer the right-hand schedule over the left-hand one. In this thesis we will formulate a cost function that takes the previously mentioned characteristics into account. Using this cost function, our aim is to obtain the cheapest schedule.

In the remainder of this chapter, a short outline of the biological background is given. After that, related work is discussed followed by a short overview on the contribution of this work. In Chapter 2 the problem is defined in a formal way. An NP-hardness result is presented in Chapter 3. In Chapter 4 existing methods are described and improved upon. New methods are introduced in Chapter 5. All the methods are evaluated experimentally in Chapter 6. Conclusions and directions for future research are given in Chapter 7.

1.1 Biological background

The *genotype* of an organism is the complete genetic information on the traits of that organism. Traits of an organism are encoded by *genes*. Genes on their turn are located on *chromosomes*. In this thesis we will only be considering *diploid* organisms. The cells of diploid organisms have two homologous copies of each chromosome. The number of chromosomes in a diploid cell is commonly denoted by $2n$, where n depends on the genetic layout of the organism in question. *Homologous chromosomes* are non-identical chromosomes that contain information for the same biological features. As such they contain the same genes at the same loci; the genetic information—called an *allele*—at those same genes may be different, though. Each member of a pair of homologous chromosomes is inherited from a different parent. In Figure 1.2 the previously introduced terms are illustrated.

A *gamete* is a *haploid* cell, i.e. the number of chromosomes in a gamete is n. Gametes are created by a biological process called *meiosis*. Meiosis takes place in special diploid cells that are called *meiocytes*. The $2n$ chromosomes are first duplicated, this results in the formation of sister-chromatid pairs. The reason why we speak of chromatids instead of chromosomes is because sister-chromatid pairs are physically connected. In the next step, tetrads are formed. *Tetrads* are structures that consist of two homologous pairs of sister-chromatids. In a tetrad, genes on non-sister chromatids can recombine, i.e. non-sister chromatids (of the same tetrad) that physically cross each other can break and reunite differently. The final phase of meiosis,



(a) Meiocyte prior to meiosis.

(b) In a tetrad recombination can occur. Sister-chromatid pairs are signaled by '{'.

(c) Observe that some recombinations have occurred (flagged by '*').

Figure 1.3: Formation of gametes proceeds in two stages: (1) during meiosis I chromosomes are duplicated and tetrads are formed, (2) in meiosis II recombination occurs and gametes are formed.

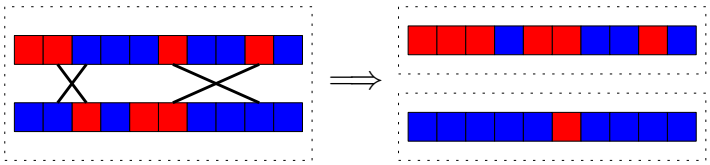


Figure 1.4: Recombination.

called meiosis II, results in the formation of four gametes. In Figure 1.3 the meiosis process is illustrated. Genotypes of gametes in which no recombination occurred are called *parental types*, otherwise they are called *recombinant types*. In Figure 1.4 the formation recombinant type gametes is shown in more detail. The fusion of two gametes results in the formation of a *zygote*. A zygote is diploid. Note that during the formation of a zygote there is no crossover. So each chromosome of a zygote originates from only one parent (cf. Figure 1.5).

Now let's look more closely into the occurrence of recombination. The *recombination frequency* (RF) is the fraction of recombinant gametes. If any two loci were to assort independently, RF would be always 0.5. However, loci that are located close to each other have a smaller chance of recombining than if they are farther apart. Using the recombination frequencies, we can describe what the probability is of an individual generating a gamete with a desired genotype. Since there are no crossover events during the formation of a zygote, the probability of obtaining an individual with a certain genotype corresponds to the product of obtaining the two gametes that resulted in that genotype. This probability can be translated into a population size in which with a certain probability the desired genotype is expected to be present.

The recombination frequency between any two loci of an organism can be measured in experimental populations [6]. A genetic map (a.k.a linkage map) is based on the correspondence that closely located loci have a small recombination frequency. The map unit of a genetic map is a *centimorgan* (cM). There can be multiple crossover events between two loci. From the

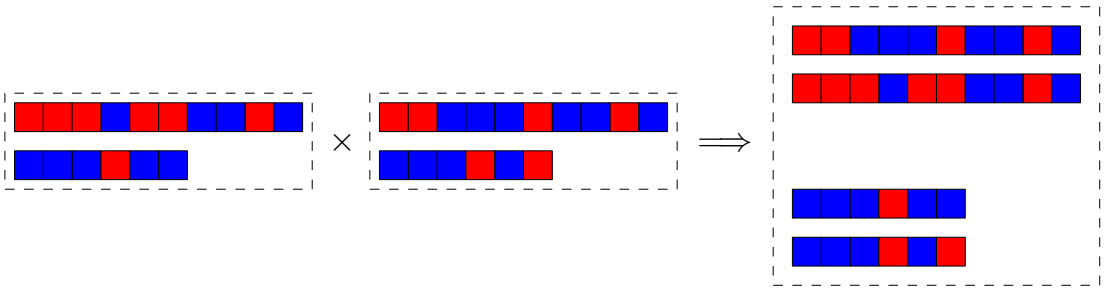


Figure 1.5: Formation of a zygote.

resulting genotypes these multiple crossover events cannot always be inferred. So RF tends to be an underestimation of the actual number of crossovers. The Haldane mapping function compensates for this and results in additive mapping distances [7]. The relation between the recombination frequency and the map distance m in Morgan is

$$m = -\frac{1}{2}\ln(1 - 2RF). \tag{1.1}$$

The recombination frequency can be obtained from a map unit as follows.

$$RF = \frac{1}{2} \cdot (1 - e^{-2m}). \tag{1.2}$$

Besides the Haldane mapping function there are other mapping functions, such as Kosambi's mapping function which also takes into account that successive crossover events inhibit each other (known as *interference*) [11].

1.2 Related work

In the plant breeding literature the process of accumulating favorable alleles into one individual is known as *gene pyramiding* [13]. In the same literature a crossing schedule is called a *gene pyramiding scheme*.

Servin et al. were the first to introduce the problem considered in this thesis in a formal way [16]. Some simplifying assumptions were made about the topology of a crossing schedule. Moreover, assumptions were made about the genotypes of the initial parents. Even though these two assumptions render their work unusable in practice, it allowed them to quantify the number of possible crossing schedules. An algorithm for generating all these crossing schedules was given. Using this algorithm, all crossing schedules for a particular case were generated. Subsequently, all the resulting crossing schedules were compared in terms of total population size needed. A useful contribution of this work is that it describes how to make use of recombination frequencies in determining the population size needed for a certain crossing. In Chapter 4 we describe Servin's work in more detail.

It is unfortunate that Servin's work has passed relatively unnoticed by the plant breeding community. Especially when considering a later paper by Ishii and Yonezawa [8], in which it is simply assumed that target genes are always unlinked (i.e. the recombination frequencies are fixed to 0.5). The work of Ishii and Yonezawa identifies the number of generations, number

of crossings and the total population size as important attributes. However, no cost function that makes use of these attributes is specified. An experimental evaluation is performed on crossing schedules having different topologies for a fixed number of parents.

Servin's and Ishii's work have in common that two stages are distinguished. First they aim to obtain a heterozygous genotype containing all target alleles. After which, using this genotype the ideotype is obtained. Ishii and Yonezawa describe this latter stage in a separate paper [9].

In order to actually perform a crossing that is described in a crossing schedule we need a way to select, out of the resulting progeny, the individual with the desired genotype. A frequently used method is called Marker Assisted Selection (MAS). Dekkers and Hospital provide a good review on this topic [5]. MAS constitutes of using easily identifiable markers instead of actual genes of interest. Useful markers have the property that they co-segregate with a gene of interest.

1.3 Our contribution

One of the directions for future research in the article of Servin et al. was to consider dynamic programming. In this thesis, we do exactly this and obtain an algorithm with a better performance.

The main part of this thesis considers a more general problem than the one introduced by Servin et al. We define this general problem in a formal way and present an NP-hardness and inapproximability result for it.

As for methods for solving the problem, we describe and analyze an existing genetic algorithm and give recommendations for further improvements. We also propose two new methods, one of which is a heuristic based on dynamic programming and the other is a randomized algorithm.

In the final part of this thesis, we perform an extensive experimental evaluation on the genetic algorithm and the two new methods.

Chapter 2

Problem Statement

In the previous chapter the relevant biological background was presented. Also the problem was introduced in an informal way. In this chapter the problem is defined more formally; we do this in Section 2.1. In Section 2.2, we present a practical example by which we illustrate the introduced definitions.

In our formulation we assume that we are dealing with only one chromosome. Furthermore, we assume that there are two homologous copies of that chromosome. For brevity's sake, we refer to a homologous chromosome as chromosome. Note that even though we assume that we are dealing with only one chromosome, the problem definition we present allows for multiple chromosomes.

2.1 Problem definition

A *genotype* is made up of two chromosomes. Each of these chromosomes is a bitstring of size m. We represent a genotype C by a $2 \times m$ matrix whose elements are either 0 or 1—these elements are called *alleles*. The first row in C, denoted C_0, is called the first chromosome of C, whereas the second row, denoted C_1, corresponds to the second chromosome. The rows of a genotype may be interchanged, e.g. $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$. Each column in C corresponds to a *locus*. So at every locus two alleles are present. A locus is said to be *homozygous* if its two alleles are identical, otherwise it is *heterozygous*. Likewise, a genotype is homozygous if all its loci are homozygous, otherwise the genotype is said to be heterozygous. The desired genotype is called the *ideotype*, which we denote by C^*.

In this problem we are not looking at genotypes only; we are also interested in how an *individual* with a particular genotype was obtained. For that purpose we define a *crossing schedule* as a *connected directed acyclic graph* (DAG) in which every node corresponds to an individual. The edges are directed towards the ideotype and relate a parent with its child. Since an individual has at most two parents, the in-degree of a node is at most 2. The out-degree of a node corresponds to the number of children the corresponding individual has. The nodes corresponding to the original parents have no incoming edges (i.e. they are *source nodes*). On the other hand, the node labeled with the ideotype has no outgoing edges (i.e. it is a *target node*), as it does not have any descendants. We refer to the individual corresponding

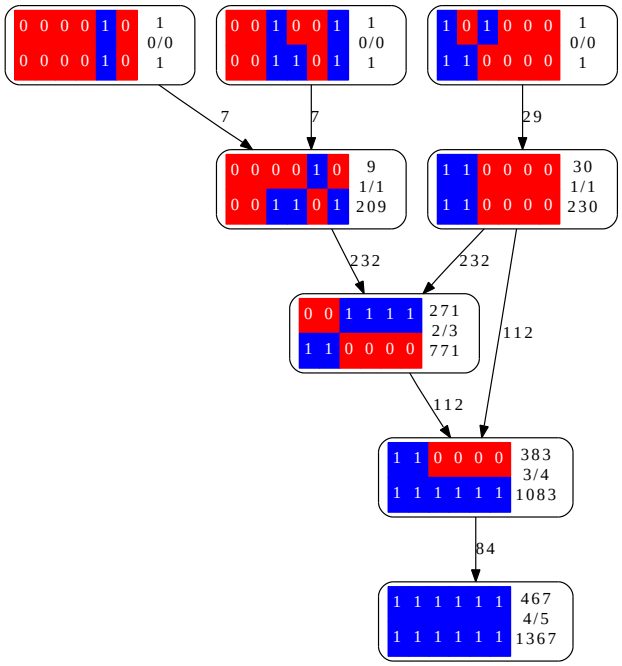


Figure 2.1: Example crossing schedule, where $cost(I) = 100 \cdot gen(I) + 100 \cdot cross(I) + pop(I)$. Per node I, from top to bottom and left to right: $pop(I)$, $gen(I)$, $cross(I)$ and $cost(I)$.

to the target node as I^*; the genotype of this node is C^*. For an individual I we are interested in the following.

1. The generation in which I was obtained, denoted by $gen(I)$.
   Corresponds to the depth of I in the crossing schedule.

2. The total population size needed, denoted by $pop(I)$.
   The exact definition of this number follows later on.

3. The number of crossings, denoted by $cross(I)$.
   This number is one bigger than the number of non-source ancestors I has in the crossing schedule.

Every individual in a crossing schedule has a cost associated with it, which indicates how expensive it is (in terms of the number of generations, the number of crossings, and the population size) to obtain that individual with the given crossing schedule. The cost function is required to be monotonically increasing in the attributes $gen(\cdot)$, $pop(\cdot)$ and $cross(\cdot)$. In Figure 2.1 a crossing schedule with a prescribed cost function is given.

Between any two loci a recombination frequency is defined. The recombination frequencies are given by the matrix RM, which is defined as follows.

Definition 2.1 *RM is an $m \times m$ matrix, whose entries are real numbers in the range $[0, 0.5]$. RM has the following properties.*

1. *All elements on* $\mathrm{diag}(RM)$ *are 0 and all other elements of* RM *are nonzero:*
   Since there can be no recombination between the same loci, we have that the diagonal of RM *contains only zeros. We also require that any two non-identical loci can recombine.*

2. $RM = RM^T$*:*
   RM *is symmetric.*

3. *For every* i, j, k*, where* $0 \leq i < j < k < m$*, it holds that* $RM_{ij} \leq RM_{ik}$*:*
   It is assumed that the loci are ordered according to their position on the genetic map. Therefore we have that the recombination frequency between loci i *and* j *is less than or equal to the recombination frequency between loci* i *and* k*.*

The next thing we have to define is how a crossing between two individuals proceeds. Let D and E be the genotypes of the two individuals that are crossed. The resulting genotype is denoted by C. Using RM, we can calculate the probability of obtaining a certain genotype out of two parental genotypes. In Section 1.1 we described that a genotype is the result of the fusion of two haploid gametes. So one of the chromosomes of the resulting genotype C corresponds to a gamete given rise to by D and the other chromosome corresponds to a gamete produced by E. We define the probability of obtaining C_0 out of D, that is $P(D \to C_0)$, as follows.

Definition 2.2 *Let* $s = (\nu(0), \ldots, \nu(k))$ *be an ordered sequence of heterozygous loci in* D*.* $P(D \to C_0)$ *is then defined as follows.*

- *If there is an allele in* C_0 *that does not occur in* D *at the same locus then* $P(D \to C_0) = 0$*;*
- *Otherwise, if* s *is empty then* $P(D \to C_0) = 1$*;*
- *Otherwise*

$$P(D \to C_0) = \frac{1}{2} \prod_{i=0}^{k-1} \begin{cases} RM_{\nu(i)\nu(i+1)}, & \text{if } (C_{0\nu(i)} = D_{0\nu(i)} \wedge C_{0\nu(i+1)} = D_{1\nu(i+1)}) \\ & \vee\ (C_{0\nu(i)} = D_{1\nu(i)} \wedge C_{0\nu(i+1)} = D_{0\nu(i+1)}). \\ 1 - RM_{\nu(i)\nu(i+1)}, & \text{otherwise.} \end{cases}$$

  The condition in the first case states that the target alleles are located on different chromosomes (so a recombination event is required to join them). The factor $1/2$ *stems from the fact that recombination results in two chromosomes one of which being the desired one.*

We can now compute the probability of obtaining C out of D and E, denoted by $P(D, E \to C)$.

Lemma 2.3

$$P(D, E \to C) = \begin{cases} P(D \to C_0) \cdot P(E \to C_1) + P(E \to C_0) \cdot P(D \to C_1), & \text{if } C_0 \neq C_1, \\ P(D \to C_0) \cdot P(E \to C_1), & \text{if } C_0 = C_1. \end{cases}$$

Proof. It holds that either C_0 is obtained from D and C_1 is obtained from E, or vice versa. Thus, we have

$$P(D, E \to C) = P(((D \to C_0) \cap (E \to C_1)) \cup ((E \to C_0) \cap (D \to C_1))).$$

Due to independence, it holds that

$$P(D \to C_0 \cap E \to C_1) = P(D \to C_0) \cdot P(E \to C_1),$$

and

$$P(D \to C_1 \cap E \to C_0) = P(D \to C_1) \cdot P(E \to C_0).$$

Recall that [12]

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

Let $A = (D \to C_0) \cap (E \to C_1)$ and $B = (E \to C_0) \cap (D \to C_1)$. If $C_0 \neq C_1$ then $P(A \cap B)$ is 0. Otherwise, we have that $P(A) = P(B) = P(A \cap B)$. The lemma now follows. □

We can relate the probability $P(D, E \to C)$ to a required population size in which at least one individual with genotype C is present with a certain probability of success γ. This population size is given by the function $N(C, D, E, \gamma)$, which is derived in the following lemma [16].

Lemma 2.4

$$N(C, D, E, \gamma) = \left\lceil \frac{\ln(1-\gamma)}{\ln(1 - P(D, E \to C))} \right\rceil.$$

Proof. The probability that none of the individuals have the right genotype C is

$$(1 - P(D, E \to C))^{N(C,D,E,\gamma)}.$$

This probability is exactly the probability of not having success, thus we have

$$(1 - P(D, E \to C))^{N(C,D,E,\gamma)} = 1 - \gamma.$$

The lemma follows if we solve for $N(C, D, E, \gamma)$. □

In Chapter 1 we mentioned that it is useful to restrict the population size per crossing. For this purpose we introduce the parameter $N_{\max}$. Only crossings for which $N(C, D, E, \gamma)$ is less than or equal to $N_{\max}$ are allowed.

The only thing we did not mention yet is how the attributes of an individual are determined. For this we need to introduce some additional notation. We denote the two parents of a non-source node I by $p_1(I)$ and $p_2(I)$. In case I is obtained via selfing, we have $p_1(I) = p_2(I)$. The genotype of an individual I is denoted by $geno(I)$. With $anc(I)$ we denote the set of ancestral individuals of I; note that I itself is also present in $anc(I)$.

The number of generations of I is defined as

$$gen(I) = \begin{cases} 0, & \text{if } I \text{ is a source,} \\ 1 + \max\{gen(p_1(I)), gen(p_2(I))\}, & \text{otherwise.} \end{cases} \tag{2.1}$$

The population size is

$$pop(I) = \sum_{I' \in anc(I)} \begin{cases} 1, & \text{if } I' \text{ is a source,} \\ N(geno(I'), geno(p_1(I')), geno(p_2(I')), \gamma), & \text{otherwise.} \end{cases} \tag{2.2}$$

The number of crossings is

$$cross(I) = \sum_{I' \in anc(I)} \begin{cases} 0, & \text{if } I' \text{ is a source,} \\ 1, & \text{otherwise.} \end{cases} \tag{2.3}$$

We now have described all the ingredients needed for defining a problem instance.

Definition 2.5 *A* problem instance *is a six tuple* $(P, C^*, RM, \gamma, N_{\max}, cost)$ *where*

- P *is the set of parental genotypes we start with, the number of parents is* n*,*
- C^* *is the ideotype,*
- RM *is the recombination matrix,*
- $\gamma \in [0, 1)$ *is the desired probability of success used in determining the population size,*
- $N_{\max} \in \mathbb{N}$ *is the maximal population size allowed per crossing,*
- *cost is a cost function that takes an individual as argument. The cost function is required to be monotonically increasing in the attributes gen, pop and cross.*

The problem now is, given a problem instance $(P, C^*, RM, \gamma, N_{\max}, cost)$, to find a crossing schedule G^* such that $cost(I^*)$ is minimal. We call this problem CrossingSchedule.

2.2 Example

The goal of this section is to illustrate the previously introduced definitions. We do this by considering a practical example that deals with a disease in pepper called powdery mildew. This disease is caused by the fungus *Leveillula Taurica*. The leaves of infected pepper plants show white powder-like spots as well as yellowish regions (such regions are called tanned lesions). In severe cases of the disease the infected pepper plant may lose a significant amount of its leaves, which in turn results in crop loss. The fungus is resistant to fungicides, so host-plant resistance is desired.

There is a pepper line that is resistant to the fungus. Three dominant QTLs that explain the resistance have been identified in this line [17]. QTLs (i.e. quantitative trait loci) are biological markers. In addition to resistance, we also look at pungency. Pungency is a dominant monogenic trait, i.e. it is explained by only one gene. The pungency gene is closely linked with one of the resistance QTLs, with a genetic distance of 0.01 cM.

The resistant line is pungent (hot). On the other hand, the elite line that is used for production is sweet but susceptible to powdery mildew. Both lines are pure lines, i.e. they are homozygous

(a) White powder-like spots. (b) Tanned lesions.

Figure 2.2: Powdery mildew in pepper.

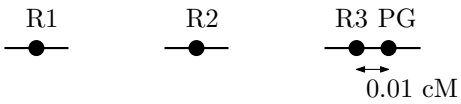


Figure 2.3: The four loci that are considered.

at all loci. The goal is to come up with a crossing schedule that results in an individual that is both resistant and sweet.

We start by formulating the corresponding problem instance of CrossingSchedule. There are four loci that we consider. The first three loci correspond to the resistance QTLs, whereas the last one corresponds to the pungency gene (cf. Figure 2.3). We denote dominant alleles with 1. Recall that both resistance and pungency are dominant traits. So the parent set is

$$P = \left\{ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \right\}.$$

The ideotype is resistant and sweet:

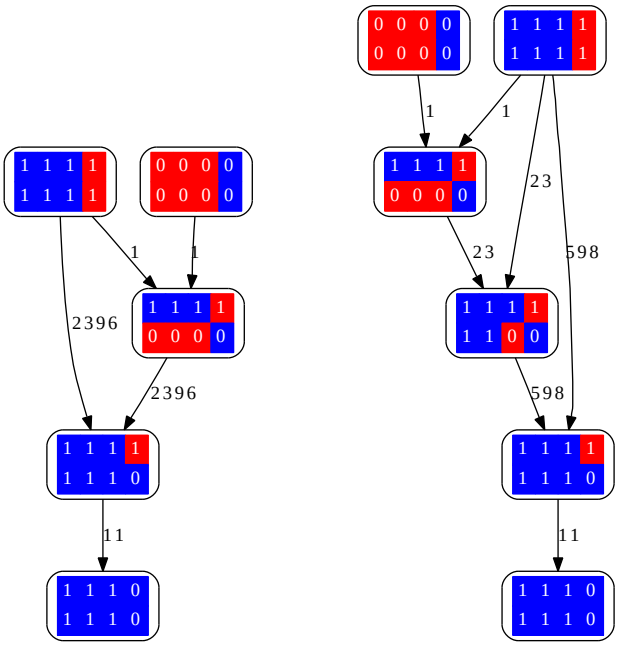
$$C^* = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

All loci are unlinked, except for the pungency locus that is linked with one of the resistance loci. In our setting this corresponds to the third and the fourth loci being linked. Using Haldane's mapping function (see Equation 1.2), we can translate the genetic distance of 0.01 cM to a recombination frequency as follows.

$$RF = \frac{1}{2} \cdot (1 - e^{-2 \cdot 0.01}) \approx 0.01.$$

Unlinked loci have a recombination frequency of 0.5. So RM is defined as

$$RM = \begin{pmatrix} 0 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0 & 0.01 \\ 0.5 & 0.5 & 0.01 & 0 \end{pmatrix}.$$



(a) Cost: 3010. (b) Cost: 1434.

Figure 2.4: Crossing schedules.

We set

$$\gamma = 0.95,$$
$$N_{\max} = 2500.$$

Finally, we choose the following cost function

$$cost(I) = 100 \cdot gen(I) + 100 \cdot cross(I) + pop(I).$$

The first crossing schedule we look at is depicted in Figure 2.4a. In the first crossing in the schedule the two parents are crossed. Since both parents are homozygous, the probability of obtaining the required genotype is 1 and consequently the population size needed is also 1.

The genotypes involved in the second crossing are

$$C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, D = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \text{ and } E = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The resulting genotype C is obtained by crossing D and E. By Lemma 2.3, we have that the probability for obtaining C out of D and E is

$$P(D, E \to C) = P(D \to C_0) \cdot P(E \to C_1) + P(E \to C_0) \cdot P(D \to C_1).$$

The second term is 0, as D cannot give rise to C_1. Since E is homozygous, we have that $P(D \to C_0) = 1$. By Definition 2.2, we have

$$P(E \to C_1) = \frac{1}{2} \cdot (1 - RM_{01}) \cdot (1 - RM_{12}) \cdot RM_{23} = \frac{1}{2} \cdot (1 - 0.5) \cdot (1 - 0.5) \cdot 0.01 = 0.00125.$$

So $P(D, E \to C) = 0.00125$. Using Lemma 2.4, we translate this probability to a population size in the following way.

$$N(C, D, E, 0.95) = \left\lceil \frac{\ln(1 - 0.95)}{\ln(1 - 0.00125)} \right\rceil = 2396.$$

The final crossing in the schedule result in the ideotype C^* by selfing genotype C. Since C^* is homozygous, we have that the probability of obtaining C^* out of C is

$$P(C, C \to C^*) = P(C \to C_0^*) \cdot P(C \to C_1^*).$$

This time we only have one heterozygous locus. Therefore we have an empty domain in the product in Definition 2.2. The probability is thus

$$P(C \to C_0^*) = \frac{1}{2}.$$

Translating this to a population size yields

$$N(C, D, E, 0.95) = \left\lceil \frac{\ln(1 - 0.95)}{\ln(1 - 0.5)} \right\rceil = 11.$$

The total population size of the schedule is 2410. The number of crossings and the number of generations is 3. So the total cost of the schedule is $100 \cdot 3 + 100 \cdot 3 + 2410 = 3010$. The most expensive crossing in this schedule is the second crossing. Besides paying for having a recombination between the last two loci, we also pay for not having recombinations among the first three loci.

In the second crossing schedule (see Figure 2.4b), we postpone the recombination between the third and fourth loci. The first and last crossing of both crossing schedules are identical. The second crossing is different; the genotypes that are involved in this second crossing are

$$C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}, D = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \text{ and } E = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The probability of obtaining C out of D and E is

$$\begin{aligned} P(D, E \to C) = P(E \to C_1) &= \frac{1}{2} \cdot (1 - RM_{01}) \cdot RM_{12} \cdot (1 - RM_{23}) \\ &= \frac{1}{2} \cdot (1 - 0.5) \cdot 0.5 \cdot (1 - 0.01) = 0.12375. \end{aligned}$$

The population size corresponding to this probability is

$$N(C, D, E, 0.95) = \left\lceil \frac{\ln(1 - 0.95)}{\ln(1 - 0.12375)} \right\rceil = 23.$$

In the next crossing we recombine the third and fourth loci by crossing C and D. The genotype that we obtain is

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

The probability of obtaining this genotype is

$$\begin{aligned} P(C, D \to B) = P(C \to B_1) &= \frac{1}{2} \cdot RM_{23} \\ &= \frac{1}{2} \cdot 0.01 = 0.005. \end{aligned}$$

Translating the probability into a population size yields

$$N(C, D, B, 0.95) = \left\lceil \frac{\ln(1 - 0.95)}{\ln(1 - 0.005)} \right\rceil = 598.$$

Now the total population size is 634. The number of crossings and the number of generations is 4. So the total cost of the schedule is $100 \cdot 4 + 100 \cdot 4 + 634 = 1434$. Therefore the second schedule is more optimal than the first one.

Chapter 3

Complexity of the Problem

In this chapter we show how the decision problem SetCover reduces to CrossingSchedule. SetCover is defined as follows.

Definition 3.1 *Given a universe U of n elements and a collection $\mathcal{S} = \{S_1, \ldots, S_l\}$ of subsets of U, a cover is a subset $\mathcal{C} \subseteq \mathcal{S}$ whose union is U. The decision problem* SetCover *is to determine, given a natural number k, whether there exists a cover $\mathcal{C}$ of cardinality at most k.*

SetCover was one of the 21 problems that were shown to be NP-complete by Karp [10]. By giving a polynomial time reduction from SetCover to our problem, we show that our problem is NP-hard.

Let $(U, \mathcal{S}, k)$, where $U = \{e_1, \ldots, e_n\}$ and $\mathcal{S} = \{S_1, \ldots, S_l\}$, be a problem instance of SetCover. The corresponding problem instance $(P, C^*, RM, \gamma, N_{\max}, cost)$ of CrossingSchedule is obtained as follows.

1. The idea is to let loci correspond to elements in the universe. Furthermore we let every subset $S \in \mathcal{S}$ correspond to a parent. So the initial set of parents P has l elements. The genotype of a parent corresponding to a subset $S \in \mathcal{S}$ is thus a $2 \times n$ matrix. The second row of this matrix contains only zeros, whereas the first row contains a 1 if the element corresponding to the locus (column index) is present in S.
2. The ideotype is an $2 \times n$ matrix whose entries are all 1.
3. RM is an $n \times n$ matrix whose diagonal contains only zeros, all other elements of RM are 0.5.
4. The desired probability of success γ is set to 0.99.
5. No restriction is put on the maximal allowed population size per crossing, i.e. $N_{\max} = \infty$.
6. The cost function corresponds to the number of crossings, i.e. $cost(I) = cross(I)$.

The reduction can be done in polynomial time (in fact in $\mathcal{O}(nl)$ time). The cost function is chosen such that the optimum crossing schedule is the one with the minimum number of crossings.

In order to show that the reduction works, we have to prove that there is a cover of cardinality at most k if and only if the cost of the optimum schedule is at most k.

($\Rightarrow$) Let $\mathcal{C} \subseteq \mathcal{S}$ be a cover such that $|\mathcal{C}| \leq k$. Recall that every subset $S \in \mathcal{S}$ corresponds to one parent. So with $|\mathcal{C}| - 1$ crossings we can obtain an individual whose genotype contains at least one 1 at every locus. The ideotype can then be obtained by a selfing step. The corresponding schedule G contains thus $|\mathcal{C}|$ crossings. Therefore, the optimum schedule G^* contains at most $|\mathcal{C}| \leq k$ crossings.

($\Leftarrow$) Let G^* be the optimum crossing schedule, and let m be the number of crossings in G^*. We have that $m \leq k$. We claim that the final crossing in G^* is a selfing step. Assume for a contradiction that this is not the case. Let I_1 and I_2 be the two distinct individuals whose crossing resulted in I^*. We either have $I_1 \notin P$ or $I_2 \notin P$, as if both I_1 and I_2 were to be in P then they would be equal (recall that the second chromosome of individuals in P has only zeros). Now assume that $I_2 \notin P$. The individual I^* can also be obtained by selfing I_1. Since $I_2 \notin P$, we require a crossing to obtain I_2 from its parents. So by obtaining I^* via a selfing of I_1, the number of crosses is at least one less than originally the case. This contradicts our assumption that G^* is optimal. Hence, the final crossing in G^* is a selfing.

All other $m - 1$ crossings involve two distinct individuals (i.e. no selfing); the argument for this is as follows. Suppose for a contradiction that there is a non-final selfing step involving an individual I in G^*. Because the selfing is non-final, the resulting individual I' participates in another crossing. Instead of using I' in this crossing, we could have used I and ended up with a schedule with fewer crossing than G^*. This would be a contradiction however.

So now we have $m - 1$ crossings, each involving two distinct individuals. Let G' be the subgraph of G^* that contains these crossings. We have that G' is connected as G^* was connected. We now want to bound the number of parents in G'. There can be at most m parents in G', as the largest number of crossings is achieved when we have a binary tree with $m - 1$ internal nodes rooted at the pre-final individual. Since parents correspond to subsets in $\mathcal{S}$ and since crossing them together resulted in the ideotype, we have that there is a cover of at most m subsets. As $m \leq k$, there is a cover of cardinality at most k.

Theorem 3.2 CrossingSchedule *is NP-hard.*

Note that the reduction preserves the approximation factor, as the number of crossings equals the number of subsets in the cover. There is an inapproximability result for SetCover: it cannot be approximated within $\mathcal{O}(\log n)$ unless P = NP [14]. Because of the approximation factor preserving reduction, this also holds for CrossingSchedule.

Theorem 3.3 *Approximating* CrossingSchedule *within* $\mathcal{O}(\log n)$ *is NP-hard.*

Chapter 4

Existing Methods and Improvements

In this chapter we look at two existing methods. We first consider, in Section 4.1, Servin's problem statement and describe and analyze the algorithm that was proposed. After which, we present a new more efficient algorithm. In Section 4.2, we look at an existing genetic algorithm. We describe in detail how the various operators have been chosen and which selection schemes are used. Subsequently, we analyze the performance of the algorithm and conclude by describing possible improvements.

4.1 Servin's problem statement

As mentioned earlier Servin et al. were the first to introduce the problem of this thesis. The authors did not consider, however, the problem in its full generality as presented in Chapter 2. Rather, they made the following two assumptions:

Assumption 4.1 *There are as many parents as there are loci, i.e.* $|P| = n = m$*. Moreover, parent* $p_i \in P$*, where* $1 \leq i \leq n$*, has two ones at locus* i *and zeros at the other loci.*

Assumption 4.2 *A crossing schedule is a binary tree, i.e. every individual participates in at most one crossing and selfing is not allowed.*

The authors assumed the ideotype to be always $\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \end{pmatrix}$. The observant reader will notice that the ideotype can never be attained without violating Assumption 4.2. That is why the process is split into two stages. First a *root genotype* is obtained, in which all the target alleles are present (but not on the same chromosome). This root genotype is then crossed with a blank parent having only zeros. The result of this crossing is $\begin{pmatrix} 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 \end{pmatrix}$. Finally, the ideotype is obtained via a selfing step. The authors do not concern themselves with these two final steps. In Figure 4.1 an example crossing schedule, including the two final steps, is given.

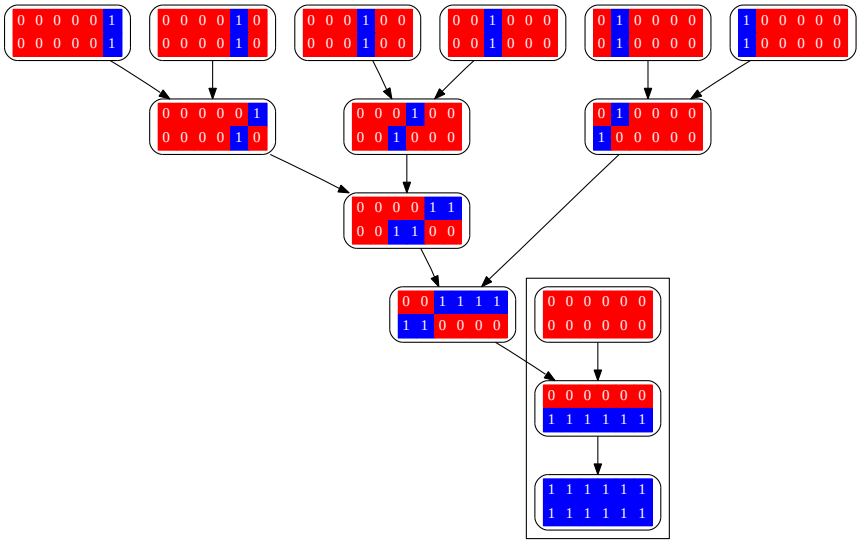


Figure 4.1: Example crossing schedule, the two final steps are depicted in the box.

4.1.1 Servin's algorithm

The first question Servin et al. pose is how many binary trees there are with n labeled leaves [15]. The root genotype is obtained by crossing two intermediary parents carrying p and $n-p$ target alleles, where $1 \leq p < n$. Every original parent contributes exactly one target allele. Therefore there are $\binom{n}{p}$ possible combinations of original parents that contribute p target alleles. If we sum we up over all possible values of p, we can compute the number of binary trees having n labeled leaves as follows.

$$A(n) = \begin{cases} 1, & \text{if } n = 1, \\ \frac{1}{2}\sum_{p=1}^{n-1} \binom{n}{p} A(p)A(n-p), & \text{if } n > 1. \end{cases} \tag{4.1}$$

In order to correct for double counting p and $n-p$, there is an additional factor of $1/2$. The recursion solves to

$$A(n) = \prod_{i=2}^{n}(2i-3) = (2n-3)!!. \tag{4.2}$$

The algorithm presented by Servin et al. performs an exhaustive enumeration and generates all possible crossing schedules. This is done by iteratively generating all trees of height h, starting from $h = 0$ up until $h = n-1$. A tree of height h is generated by merging two subtrees, one of height $h-1$ and one of height at most $h-1$. Only subtrees that are disjoint are merged, as otherwise the result would not be a tree. Merged trees that have n leaves are stored. In Algorithm 1 the pseudocode of this procedure is given.

Unfortunately, the running time of the algorithm is not given. So we have to derive this ourselves. We start by bounding $|\mathcal{S}_{n,h}|$—the cardinality of the set of all trees of height h having at most n leaves. The height h of a binary tree with k leaves is bounded by $\lceil \lg k \rceil \leq h \leq k-1$. Conversely, the number of leaves is bounded by $h+1 \leq k \leq 2^h$. So in our case, where the

Algorithm 1: GENERATE(P)

Input: P is the set of original parents adhering to Assumption 4.1
Result: All possible trees resulting in a root genotype given P

```
1  n ← |P|
2  L ← ∅
3  S_{n,0} ← P
4  for h ← 1 to n − 1 do
5    foreach G_1 ∈ S_{n,h−1} do
6      foreach G_2 ∈ ⋃_{i=0}^{h−1} S_{n,i} do
7        if G_1 and G_2 are disjoint then
8          G ← MERGE(G_1, G_2)
9          S_{n,h} ← S_{n,h} ∪ {G}
10         if G has n leaves then
11           L ← L ∪ {G}
12 return L
```

number of leaves is at most n, we can say that

$$
\begin{aligned}
|\mathcal{S}_{n,h}| &= \sum_{i=h+1}^{\min\{2^h,n\}} \binom{n}{i} (2i-3)!! && \text{(equation 4.2, } \tbinom{n}{i} \text{ ways to pick } i \text{ leaves)} \\
&\leq \sum_{i=h+1}^{n} \binom{n}{i} (2i)!! && (2^h \leq n \text{ and } (2i-3)!! \leq (2i)!!) \\
&\leq \sum_{i=h+1}^{n} \frac{n^i\, 2^i\, i!}{i!} && (\tbinom{n}{i} \leq \tfrac{n^i}{i!} \text{ and } (2i)!! = 2^i\, i!) \\
&\leq \sum_{i=h+1}^{n} (2n)^i.
\end{aligned}
$$

We now define $a_i = (2n)^i$, and reverse the order of the terms in the summation as follows.

$$
|\mathcal{S}_{n,h}| \leq \sum_{i=h+1}^{n} a_i = \sum_{i=0}^{n-(h+1)} a_{n-i}.
$$

The summation in the previous equation can be bounded by an infinitely decreasing geometric series, as the ratio of two consecutive terms is bounded by

$$
\begin{aligned}
\frac{a_{n-(i+1)}}{a_{n-i}} &= \frac{(2n)^{n-(i+1)}}{(2n)^{n-i}} && (a_i = (2n)^i) \\
&= \frac{1}{2n} \\
&\leq \frac{1}{2}. && (n > 0)
\end{aligned}
$$

Thus we can conclude that

$$|\mathcal{S}_{n,h}| \leq \sum_{i=h+1}^{n} a_i \leq \sum_{i=0}^{\infty} a_n \left(\frac{1}{2}\right)^i = a_n \frac{1}{1-1/2} = 2n^n\, 2^n. \tag{4.3}$$

We proceed with bounding $|\bigcup_{i=0}^{h} \mathcal{S}_{n,i}|$ as follows.

$$|\bigcup_{i=0}^{h} \mathcal{S}_{n,i}| = n + \sum_{i=1}^{h} |\mathcal{S}_{n,i}| \leq n + 2hn^n\, 2^n. \tag{4.4}$$

The check in line 7 of Algorithm 1 requires $\mathcal{O}(n)$ time. We assume that all the other operations require constant time. The running time is now as follows.

$$\begin{aligned}
\sum_{h=1}^{n-1} |\mathcal{S}_{n,h-1}| \cdot |\bigcup_{i=0}^{h} \mathcal{S}_{n,i}| \cdot \mathcal{O}(n) &\leq \mathcal{O}(n) \sum_{h=1}^{n-1} (2n^n\, 2^n) \cdot (n + 2hn^n\, 2^n) \quad \text{(equations 4.3 and 4.4)} \\
&= \mathcal{O}(n^{n+1}\, 2^n) \sum_{h=1}^{n-1} \mathcal{O}(hn^n\, 2^n) \\
&= \mathcal{O}(n^{2n+1}\, 2^{2n}) \sum_{h=1}^{n-1} \mathcal{O}(h) \\
&= \mathcal{O}(n^{2n+3}\, 2^{2n}). \qquad (\textstyle\sum_{h=1}^{n-1} \mathcal{O}(h) = \mathcal{O}(n^2))
\end{aligned}$$

The storage of $\mathcal{S}$ dominates the space consumption; the size of $\mathcal{S}$ follows directly from Equation 4.4. The following theorem summarizes the results of this section.

Theorem 4.3 *Servin's method requires $\mathcal{O}(n^{2n+3}\, 2^{2n})$ time and consumes $\mathcal{O}(n^{n+1}\, 2^n)$ space.*

From the running time it can be seen that the considered method only works for small values of n. Servin et al. conclude their paper by stating that future research should focus on finding a dynamic programming algorithm. In the next subsection we do exactly this.

4.1.2 Dynamic programming algorithm

We start by introducing some additional notation. We denote the minimal set of parents containing all target alleles of chromosome C_0 by $A(C_0)$. The question that needs to be answered first is what kind of genotypes occur in a crossing schedule under the restrictions of Assumptions 4.1 and 4.2. In the following definition such genotypes are defined.

Definition 4.4 *A genotype C is* valid *if*

1. *there is at least one heterozygous locus in C, i.e. $A(C_0) \neq A(C_1)$, and*
2. *all homozygous loci in C have two zero alleles, i.e. $A(C_0) \cap A(C_1) = \emptyset$.*

We claim that only valid genotypes occur in the inner nodes of a crossing schedule. The proof of this is as follows.

Lemma 4.5 *In a crossing schedule adhering to Assumptions 4.1 and 4.2, all genotypes present in the inner nodes of that schedule are valid.*

Proof. Let $\mathcal{T}$ be a crossing schedule. By Assumption 4.2, we have that $\mathcal{T}$ is a binary tree. We prove the two statements of Definition 4.4 separately.

Assume for a contradiction, that there is an inner node in $\mathcal{T}$ whose genotype D is homozygous. Since $\mathcal{T}$ is a binary tree, the two chromosomes of D must originate from two disjoint subtrees. Since $D_0 = D_1$ and by Assumption 4.1, the same set of parents is needed for obtaining D_0 and D_1. This means that $\mathcal{T}$ is not a tree and we have a contradiction.

As for the second statement, assume there is an inner node in $\mathcal{T}$ whose genotype E contains a homozygous locus at index i with two one alleles. Since $\mathcal{T}$ is a binary tree, the two chromosomes of E must originate from two disjoint subtrees. The subtrees, however, cannot be disjoint, as by Assumption 4.1, both of them must contain parent p_i. Again, we have a contradiction. The lemma now follows. □

The next thing we have to do is to characterize the cost and structure of an optimal solution. Just like Servin does, we only consider the population size in the cost function, i.e.

$$cost(I) = \begin{cases} 1, & \text{if } I \text{ is a leaf,} \\ cost(I_1) + cost(I_2) + N(I, I_1, I_2, \gamma), & \text{if } I \text{ is an inner node, with children } I_1 \text{ and } I_2. \end{cases} \tag{4.5}$$

A solution is a tree whose root is labeled with a root genotype, i.e. a genotype that contains all target alleles. A tree consists of subtrees. We claim that an optimal tree consists of optimal subtrees. Before we can prove this, we have to take a closer look at the leaves present in a subtree. Because of Assumptions 4.1 and 4.2, a crossing schedule resulting in a root genotype uses all n parents and therefore has n leaves. Let's look at an intermediary genotype C. The leaves of the subtree rooted at C are exactly the parents providing the target alleles present in C. If more parents were used than required, the root genotype could never have been attained without violating Assumption 4.2. This proves the following lemma.

Lemma 4.6 *In a crossing schedule, resulting in a root genotype and adhering to Assumptions 4.1 and 4.2, the leaves of the subtree rooted at any node with a genotype C are $A(C_0) \cup A(C_1)$.*

The following corollary follows directly from the previous lemma.

Corollary 4.7 *Let $\mathcal{T}$ be a crossing schedule resulting in a root genotype and adhering to Assumptions 4.1 and 4.2. For any node in $\mathcal{T}$ it holds that both of the chromosomes of its genotype contain at least one target allele.*

The reason that we do not consider the number of crossings in the cost function is as follows. The number of crossings of an individual I corresponds to the number of inner nodes in the subtree rooted at I. In a binary tree the number of inner nodes is one less the number of leaves. By Lemma 4.6, we have that the set of leaves is fixed given a valid genotype. Therefore the number of crossings is fixed as well. Hence, there is no point in including this number in the cost function. If the number of generations were to be included in the cost function then the following lemma would not hold.

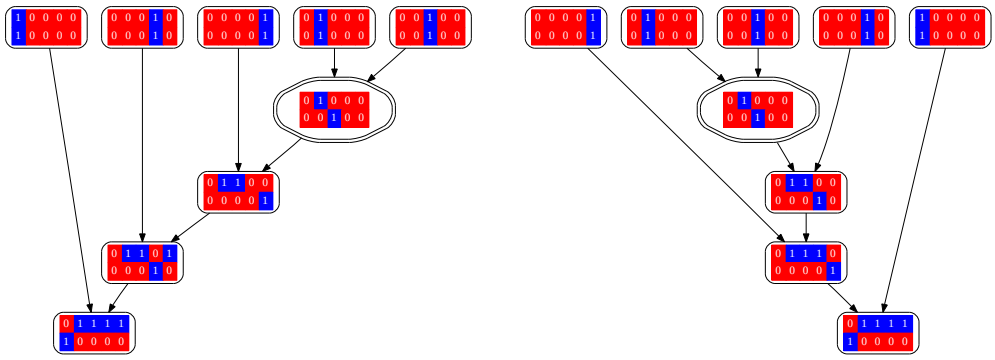


Figure 4.2: Overlapping subproblems: the genotype in the hexagon is used in two different subproblems.

Lemma 4.8 *Let $\mathcal{T}^*$ be an optimal crossing schedule resulting in a root genotype and adhering to Assumptions 4.1 and 4.2. Any node in $\mathcal{T}^*$ must be optimal for the genotype by which it is labeled.*

Proof. Consider any node I in $\mathcal{T}^*$, denote its genotype by C. First, we consider the case where I is a leaf. By Assumption 4.1, a leaf corresponds to an original parent in P; say that I corresponds to parent p_i (where $1 \leq i \leq n$). Parent p_i is homozygous at locus i. By Lemma 4.5, there can never an inner node labeled by C. So there is only one way to obtain C, hence I is optimal.

Now we look at the case where I is an inner node of $\mathcal{T}^*$. Let I_1 and I_2 be the two children of I, we denote their genotypes by D and E, respectively. We want to prove that if I is optimal for C then I_1 and I_2 must be optimal for D and E, respectively. We proof the contrapositive of this statement. The following two cases can be distinguished.

- I_1 is not optimal for D.

  We have to show that I can be obtained more optimally. Let I_1' be the root node of a subtree that results in D optimally. By Lemma 4.6, we have that the leaves of I_1' and I_1 are the same. The subtree rooted at I_1' is disjoint from the subtree rooted I_2, as by Lemma 4.5 we have that $A(I_1) \cap A(I_2) = \emptyset$. Therefore we can use I_1' instead of I_1 without violating Assumption 4.2. If we do so, we obtain I more optimally, as $cost(I_1') < cost(I_1)$.

- I_2 is not optimal for E.

  This case is analogous to the previous one.

The lemma now follows. □

The previous lemma tells us that the problem of finding a crossing schedule resulting in a root genotype has optimal substructure. The problem also exhibits overlapping subproblems (cf. Figure 4.2). Therefore dynamic programming is applicable [4].

$$f\left(\begin{pmatrix}1 & 1 & 1 & 0\end{pmatrix}\right) = \left\{\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 1 & 0\end{pmatrix}, \begin{pmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\end{pmatrix}, \begin{pmatrix}1 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\right\}.$$

Figure 4.3: Example of $f(C_0)$.

We now proceed with defining the cost of an optimal solution in terms of the optimal solutions to subproblems. Before we can do so, we must first identify the set of genotypes that can result in a certain chromosome C_0. We denote this set by $f(C_0)$. If C_0 has only one target allele then $f(C_0)$ is a singleton containing the parent providing that target allele. Otherwise, $f(C_0)$ is the set of *valid genotypes* D such that

1. D contains only the target alleles of C_0, i.e. $A(D_0) \cup A(D_1) = A(C_0)$, and
2. both of the chromosomes of D contain at least one target allele of C_0, i.e. $\emptyset \subset A(D_0)$ and $\emptyset \subset A(D_1)$.

The first case is by Lemma 4.6 and the latter is by Corollary 4.7. The cardinality of $f(C_0)$ is

$$|f(C_0)| = \begin{cases}1, & \text{if } |A(C_0)| = 1,\\ 2^{|A(C_0)|-1} - 1, & \text{if } |A(C_0)| > 1.\end{cases} \tag{4.6}$$

The first case of the previous equation is trivial. Let's look at the latter case. A valid genotype resulting in C_0 can only contain the target alleles present in C_0. Furthermore, such a genotype must contain at least one target allele but not all of them. Therefore there are $2^{|A(C_0)|} - 2$ valid genotypes that can result in C_0. Since the chromosomes of a genotype may be interchanged, we multiply by an additional factor of $1/2$. In Figure 4.3 an example is given.

Let $M[C_0, C_1]$ be the minimum cost required for obtaining C, where C is a valid genotype or an original parent. $M[C_0, C_1]$ is defined as follows.

$$M[C_0, C_1] = \begin{cases}1, & \text{if } C \in P,\\ \min\limits_{D \in f(C_0), E \in f(C_1)} \{N(C, D, E, \gamma) + M[D_0, D_1] + M[E_0, E_1]\}, & \text{otherwise.}\end{cases} \tag{4.7}$$

We store M as a two dimensional array A. Only the upper triangle of A is used, as A is symmetric. We index A using a bijective map g that maps any chromosome whose number of target alleles k is $1 \leq k < n$ to a natural number in the range $[1, 2^n - 2]$ such that for any C_0 and C_1, if $|A(C_0)| < |A(C_1)|$ then $g(C_0) < g(C_1)$. So the first n numbers g maps to are chromosomes with one target allele, the next $\binom{n}{2}$ numbers are chromosomes with two target alleles, etc. Since g is bijective, its inverse g^{-1} is also defined. In Figure 4.4 the dynamic programming table is given for the case where $n = 4$.

Suppose that the number of target alleles in a valid genotype C is $k > 1$. In order to compute the optimal cost of C, all valid genotypes with at most $k - 1$ target alleles must have already been computed. This is established, if we fill A column per column starting from the leftmost one. In Algorithm 2 the pseudocode that does exactly this is given.

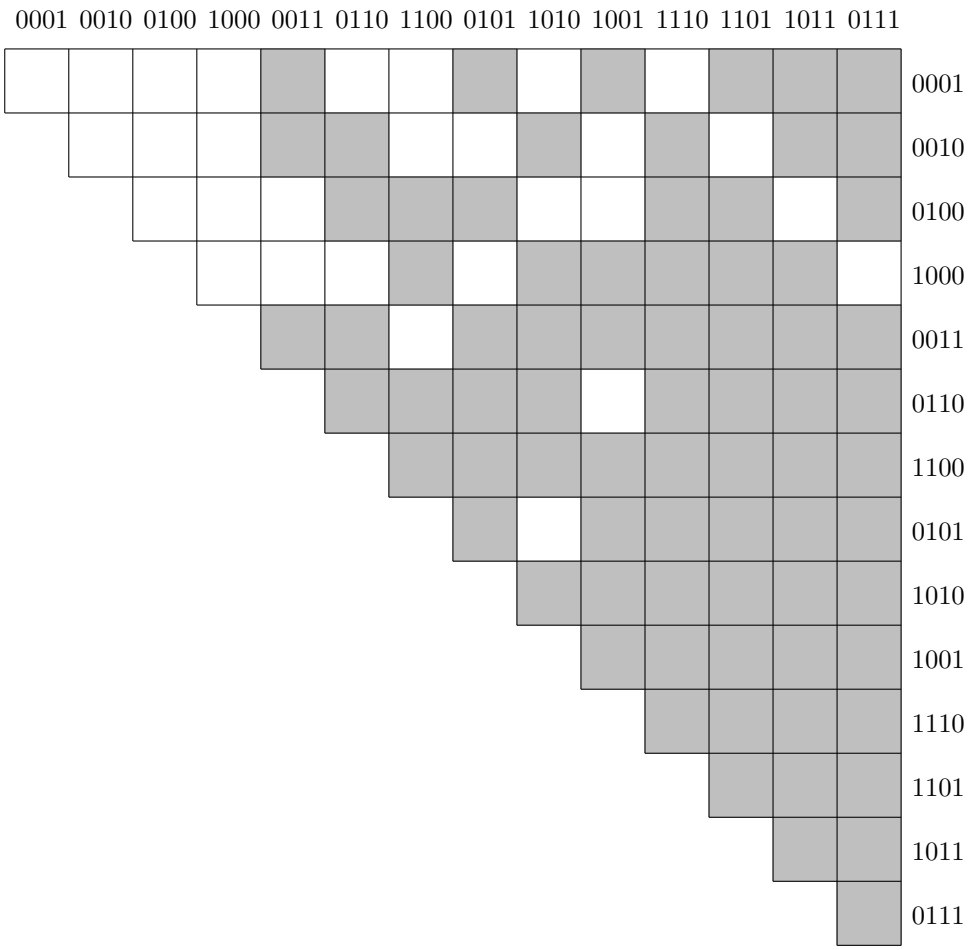


Figure 4.4: Dynamic programming table, gray entries represent invalid genotypes.

Algorithm 2: ServinDP(P, γ)

Input: P is the set of original parents adhering to Assumption 4.1
Result: the dynamic programming table

```
1  n ← |P|
2  foreach C ∈ P do
3    A[g(C_0), g(C_1)] ← 1
4    A[g(C_0), g(C_1)].p_1 ← nil
5    A[g(C_0), g(C_1)].p_2 ← nil
6  for i ← 1 to 2^n − 2 do
7    for j ← i to 2^n − 2 do
8      C_0 ← g^{-1}(i)
9      C_1 ← g^{-1}(j)
10     if C is valid then
11       A[i, j] ← ∞
12       foreach D ∈ f(C_0) do
13         foreach E ∈ f(C_1) do
14           if A[i, j] > N(C, D, E, γ) + A[g(D_0), g(D_1)] + A[g(E_0), g(E_1)] then
15             A[i, j] ← N(C, D, E, γ) + A[g(D_0), g(D_1)] + A[g(E_0), g(E_1)]
16             A[i, j].p_1 ← D
17             A[i, j].p_2 ← E
18 return A
```

Whenever we have determined the optimal cost of an entry in A, we store pointers to the two parents of that entry to allow for easy reconstruction of the crossing schedule. Recall that two more additional steps need to be performed in order to end up with the ideotype (crossing with a blank parent and a selfing). There are 2^{n-1} root genotypes. We have to examine all of these in order to determine the best overall crossing schedule.

The only thing that remains to be done is to determine the running time of ServinDP. In order to do this, we have to bound the number of genotypes with a fixed number of target alleles. Suppose that we are looking at a chromosome C_0 with k target alleles. For C_0 to comprise a valid genotype C, it must hold that the other chromosome C_1 does not contain the k target alleles of C_0. Furthermore, by Corollary 4.7, C_1 must contain at least one target allele. Therefore there are $2^{n-k}-1$ chromosomes that can be paired up with C_0. The number of chromosomes with k target alleles is $\binom{n}{k}$. So the number of valid genotypes with $k+1$ target alleles is at most

$$\binom{n}{k}(2^{n-k}-1).$$

In lines 12-17 of Algorithm 2 all pairs in $f(C_0) \times f(C_1)$ are considered. The cardinality of this Cartesian product is maximal when $|A(C_0)| = |A(C_1)|$, in which case we have

$$\begin{aligned} |f(C_0) \times f(C_1)| &= \left(2^{\frac{k+1}{2}-1}-1\right)^2 && (C \text{ has } k+1 \text{ target alleles}) \\ &= \left(2^{\frac{k-1}{2}}-1\right)^2 \\ &< 2^k. \end{aligned}$$

We can now bound the number of times lines 12-17 are executed by

$$\begin{aligned}\sum_{k=1}^{n-1}\binom{n}{k}(2^{n-k}-1)\cdot 2^k &< \sum_{k=1}^{n-1}\binom{n}{k}2^n \\ &< 2^n\sum_{k=0}^{n}\binom{n}{k} \\ &= 2^{2n}. && \text{(binomial expansion)}\end{aligned}$$

The computation of $N(C, D, E, \gamma)$ requires $\mathcal{O}(n)$ time. So lines 12-17 require $\mathcal{O}(n4^n)$ time in total. Determining whether a genotype is valid also requires linear time. This is done $\mathcal{O}(4^n)$ times. So without considering lines 12-17 we also require $\mathcal{O}(n4^n)$ time. The space consumption follows from the size of A.

Theorem 4.9 *Using dynamic programming, Servin's problem can be solved in $\mathcal{O}(n4^n)$ time and $\mathcal{O}(4^n)$ space.*

Judging from Figure 4.4, it seems that A is a sparse matrix. This is indeed the case, as the number of genotypes in A is

$$\frac{1}{2}\sum_{k=1}^{n-1}\binom{n}{k}(2^{n-k}-1)+n. \tag{4.8}$$

Since A is symmetric, we multiply by $1/2$. There is also an additional term of n, as A is initialized with the original parents in P and none of these genotypes is valid. The previous equation solves to

$$\begin{aligned}&\frac{1}{2}\sum_{k=1}^{n-1}\binom{n}{k}(2^{n-k}-1)+n \\ &= \frac{1}{2}\left[\sum_{k=1}^{n-1}\binom{n}{k}2^{n-k}-\sum_{k=1}^{n-1}\binom{n}{k}\right]+n \\ &= \frac{1}{2}\left[\left(\sum_{k=0}^{n}\binom{n}{k}2^{n-k}-2^n-1\right)-\left(\sum_{k=0}^{n}\binom{n}{k}-2\right)\right]+n \\ &= \frac{1}{2}\left[(3^n-2^n-1)-(2^n-2)\right]+n && \text{(binomial expansion)} \\ &= \frac{1}{2}[3^n-2^{n+1}+1]+n.\end{aligned}$$

If we make use of perfect hashing, we can represent M more efficiently by only storing a valid genotype once its cost has been computed. This would result in a more space efficient algorithm requiring only $\mathcal{O}(3^n)$ space.

Theorem 4.10 *Using dynamic programming with a hash table, Servin's problem can be solved in $\mathcal{O}(n4^n)$ time and $\mathcal{O}(3^n)$ space.*

4.2 Genetic algorithm

Due to confidentiality, the exact contents of this section have been omitted. In the non-public version of this thesis, we have described in this section the genetic algorithm as it has been implemented within Keygene N.V. Subsequently, we have analyzed the performance of the algorithm and presented some recommendations. We have also discussed that for this problem it is very difficult to find an effective crossover operator. Therefore we consider in Chapter 5 alternative methods.

Chapter 5

New Methods

In this chapter we describe two new methods for solving the problem introduced in Chapter 2. In Section 5.1, we present a heuristic based on dynamic programming together with a detailed description on the steps that were taken to obtain it. In the subsequent section, Section 5.2, a randomized algorithm is described.

The cost function that is used throughout this chapter is restricted to a linear combination of the number of generations, the number of crossings and the population size. The coefficients are required to be positive real numbers and at least one of them must be bigger than zero, i.e.

$$cost(gen, cross, pop) = a \cdot gen + b \cdot cross + c \cdot pop,$$

where $a, b, c \in \mathbb{R}^+$ and it holds that either $a > 0$, or $b > 0$, or $c > 0$.

5.1 Dynamic programming heuristic

In this section we start by describing a dynamic programming algorithm. This algorithm is refined in a stepwise fashion, ultimately resulting in a polynomial-time heuristic. All the steps taken are described in this section.

5.1.1 Dynamic programming

The set of genotypes that can result in a certain chromosome C_0 is denoted by $f(C_0)$. Using this function, we define a recurrence relation $M(i, C)$ that specifies a cost of obtaining genotype C in at most i generations. In order to simplify the discussion, we will view M as a table. Every entry in a M defines a crossing schedule. Besides the cost, we store for an entry $M[i, C]$:

- the genotypes of the two parents out of which C is obtained in at most i generations, denoted by $p_1(i, C)$ and $p_2(i, C)$,
- the set of ancestors $\text{ANC}(i, C)$ containing pairs (j, B) where B is an ancestor of C present at generation $j < i$ (C itself is present at generation at most i), and
- the number of generations needed to obtain C, denoted by $\text{GEN}(i, C)$.

In case a genotype C is obtained by selfing, we have $p_1(i, C) = p_2(i, C)$. Furthermore, it holds that $\textsc{gen}(i, C) \leq i$. We define the cost $M[i, C]$ as follows.

$$M[i,C] = \begin{cases} cost(0,0,1), & \text{if } i = 0 \text{ and } C \in P, \\ \infty, & \text{if } i = 0 \text{ and } C \notin P, \\ \min\limits_{\substack{D \in f(C_0) \\ E \in f(C_1)}} \Big\{ cost\big(gen(i-1,D,E), cross(i-1,D,E), & \\ \qquad pop(i-1,C,D,E)\big), M[i-1,C] \Big\}, & \text{if } i > 0, \end{cases}$$

where

$$\begin{aligned} gen(i,D,E) &= 1 + \max\big\{\textsc{gen}(i,D), \textsc{gen}(i,E)\big\}, \\ cross(i,D,E) &= 1 + \big|\big\{(j,B) \in anc(i,D,E) \mid B \notin P\big\}\big|, \\ pop(i,C,D,E) &= N(C,D,E,\gamma) \\ &\quad + \sum_{(j,B) \in anc(i,D,E)} \begin{cases} 1, & \text{if } B \in P, \\ N(B, p_1(j,B), p_2(j,B), \gamma), & \text{if } B \notin P. \end{cases} \end{aligned}$$

It can be the case that the crossing schedules defined by $M[i, D]$ and $M[i, E]$ have a genotype in common, possibly even at different generations. Only the genotype that occurs the first, i.e. has the smallest generation, should be considered in computing $pop(i, D, E)$ and $cross(i, D, E)$. That is why we define the set $anc(i, D, E)$ as a subset of the union of $\textsc{anc}(i, D)$ and $\textsc{anc}(i, E)$. If in both $\textsc{anc}(i, D)$ and $\textsc{anc}(i, E)$ a genotype B occurs then only the one with the smallest generation is present in $anc(i, D, E)$. Recall that the number of crosses in a crossing schedule is equal to the number of non-source nodes. In our current setting this number can be determined by counting the number of non-parental genotypes in $anc(i, D, E)$. The population size is also defined in terms of $anc(i, D, E)$.

Using $p_1(\cdot)$ and $p_2(\cdot)$ recursively, we can reconstruct the crossing schedule that results in any genotype C in at most i generations, provided that $M[i, C] \neq \infty$. We can see in the recurrence that $M[i, C]$ depends only on entries whose generation is less than i. Therefore we can fill M in a bottom-up fashion. In Figure 5.1 an example of M is given.

To bound the time needed for computing an entry, we need to determine the cardinality of $f(C_0)$. The cardinality can be obtained by considering all possible bipartitions of the alleles in C_0. Consider such a bipartition Q where the number of alleles in the first part is k and in the second part $m - k$. A genotype D adheres to Q if one of its chromosomes contains the k alleles of the first part, while at the same time its other chromosome contains the $m - k$ alleles of the second part. The number of genotypes adhering to Q is $2^k \cdot 2^{m-k}$. On the other hand, the number of partitions comprised by two parts containing k and $m - k$ alleles each is $\binom{m}{k}$. If we consider all possible bipartitions, we have

$$\frac{1}{2} \sum_{k=0}^{m} \binom{m}{k} 2^k 2^{m-k}.$$

As the chromosomes of a genotype may be interchanged, we multiply by an additional factor of $1/2$. By the binomial theorem, we have that the previous equation equals

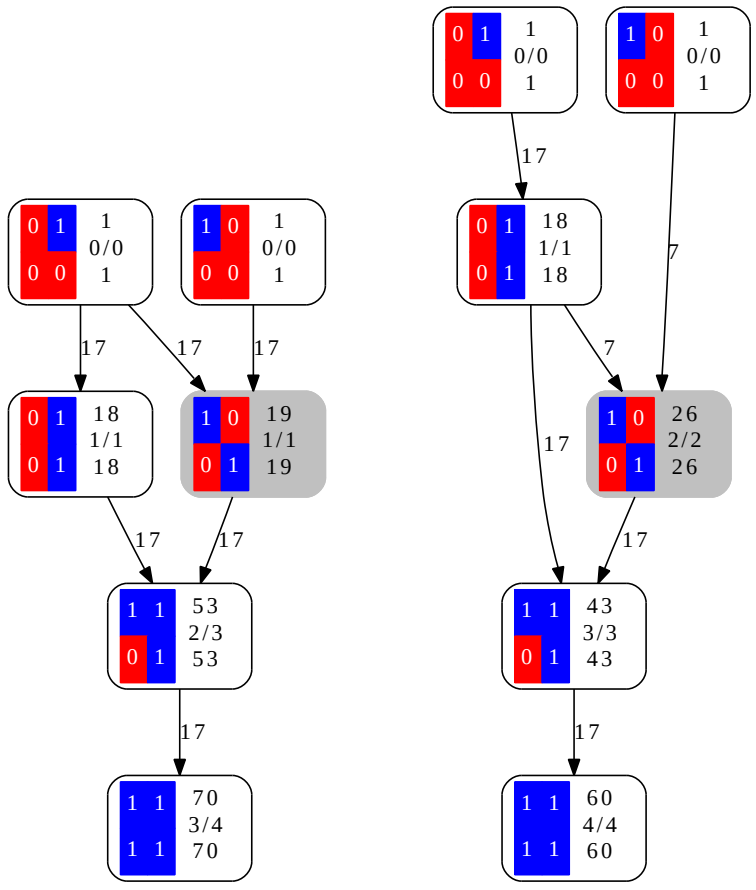
$$2^{2m-1}.$$

4	18	1	1	53	18	18	19	53	53	70
3	18	1	1	53	18	18	19	53	53	70
2	18	1	1	53	18	18	19	53	53	91
1	18	1	1	∞	18	18	19	∞	∞	∞
0	∞	1	1	∞	∞	∞	∞	∞	∞	∞
	00 00	01 00	10 00	11 00	01 01	10 10	10 01	11 10	11 01	11 11

(a) Dynamic programming table M, where $m = 2$ and $g = 5$.

$$
\begin{aligned}
P &= \left\{ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \right\} \\
C^* &= \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \\
RM &= \begin{pmatrix} 0 & 0.5 \\ 0.5 & 0 \end{pmatrix} \\
\gamma &= 0.99 \\
N_{\max} &= \infty \\
cost(g, c, p) &= p
\end{aligned}
$$

(b) Problem instance used; only the population size is considered.



(c) Crossing schedule corresponding to M.

(d) More optimal crossing schedule. Observe that while the genotype colored in gray is obtained more expensively, its descendants are cheaper.

Figure 5.1: Dynamic programming example.

So the cardinality of $f(C_0)$ is $\mathcal{O}(2^{2m})$. In order to compute $M[i, C]$, we need to look at all pairs in $f(C_0) \times f(C_1)$. The number of such pairs is $\mathcal{O}(2^{4m})$. Computing $M[i, C]$ also requires the computation of $anc(i, D, E)$. The latter can be obtained by sorting the union of $\text{ANC}(i, D)$ and $\text{ANC}(i, E)$. In the worst case the number of ancestors for an entry at generation i is 2^i. So computing $anc(i, D, E)$ requires $\mathcal{O}(2^i \log 2^i) = \mathcal{O}(i2^i)$ time. On the other hand, determining $N(C, D, E, \gamma)$ requires $\mathcal{O}(m)$ time. The number of rows in M is g and every row has $\mathcal{O}(2^{2m})$ entries. Therefore, we can conclude that the time required for filling M is

$$\sum_{i=1}^{g} \mathcal{O}((i2^i + m)2^{2m}2^{4m}) = \sum_{i=1}^{g} \mathcal{O}((i2^i + m)2^{6m}).$$

Since $\sum_{i=1}^{g} i2^i$ can be bounded by an infinitely decreasing geometric series, we have that the previous equation equals

$$\mathcal{O}((2^g + m)g2^{6m}).$$

Contrary to Servin's problem (see Section 4.1), the problem we are considering does not exhibit optimal substructure. Therefore the just described method is not an exact one. There is no optimal substructure because the subgraphs that are considered in the cost function are not used independently. Indeed, in computing the population size and the number of crosses of two subgraphs combined, the part they have in common is considered only once. So it can be the case that two *suboptimal* subgraphs may combined lead to a crossing schedule being more optimal than if two *optimal* subgraphs were to be combined instead. In Figure 5.1d an example of such a case is given.

Storing all the ancestors requires $\sum_{i=1}^{g} \mathcal{O}(2^i 2^{2m}) = \mathcal{O}(2^g 2^{2m})$ space, this dominates the storage of M. The following theorem summarizes the result we have thus far.

Theorem 5.1 *The dynamic-programming algorithm described above results in a heuristic with a running time of* $\mathcal{O}((2^g + m)g2^{6m})$ *and space consumption of* $\mathcal{O}(2^g 2^{2m})$.

5.1.2 Restricting genotypes

Instead of storing all generations, as is the case with M, we only store the last generation considered so far in the table A. Doing so makes the size of the table $\mathcal{O}(4^m)$ instead of $\mathcal{O}(g4^m)$. The cost of obtaining genotype C is given by $A[C]$. Similarly to M, we also store

- the genotypes of the two parents out of which C is obtained, denoted by $p_1(C)$ and $p_2(C)$,
- the set of ancestors $\text{ANC}(C)$ containing the ancestral genotypes of C, and
- the number of generations needed to obtain C, denoted by $\text{GEN}(C)$.

Initially the cost and the attributes are as follows.

$$A[C] = \begin{cases} cost(1, 0, 0), & \text{if } C \in P, \\ \infty, & \text{if } C \notin P, \end{cases}$$
$$\text{ANC}(C) = \emptyset,$$
$$\text{GEN}(C) = 0.$$

In the previous subsection we computed the cost of a genotype C in a backward way by making use of $f(C_0)$ and $f(C_1)$. This time, however, we work in a forward manner. We do this by maintaining two list $\mathcal{L}_{\text{old}}$ and $\mathcal{L}_{\text{cur}}$. The latter contains the genotypes that have been altered in the previous iteration, whereas the first list contains older genotypes. Initially, $\mathcal{L}_{\text{old}}$ is empty and $\mathcal{L}_{\text{cur}}$ is comprised by the parents in P. During an iteration all crossings are made among the genotypes in $\mathcal{L}_{\text{cur}}$. Similarly, all crossings are made between the genotypes in $\mathcal{L}_{\text{old}}$ and $\mathcal{L}_{\text{cur}}$. The resulting genotypes of all the crossings are stored in $\mathcal{L}_{\text{new}}$. Only the top k genotypes in $\mathcal{L}_{\text{new}}$ are retained; later in this subsection we will describe how this is done. After all crossings have been performed, the contents of $\mathcal{L}_{\text{cur}}$ are put into $\mathcal{L}_{\text{old}}$ and $\mathcal{L}_{\text{new}}$ becomes $\mathcal{L}_{\text{cur}}$. The final step in an iteration is to commit the genotypes in $\mathcal{L}_{\text{new}}$ into A. This requires an update of the attributes $\textsc{gen}(C)$ and $\textsc{anc}(C)$ for every genotype C in A. The number of iterations that is performed is g.

Let's look at how a crossing between two genotypes D and E proceeds. Let the number of heterozygous loci in D and E be denoted by k and l, respectively. The number of gametes that D can produce is 2^k. Similarly, E can give rise to 2^l different gametes. So the number of genotypes that can result is at most 2^{k+l}. Every one of these genotypes is considered. Let C be such a genotype. The cost of obtaining C out of D and E is given by

$$\mathit{cost}\big(\mathit{gen}(D,E), \mathit{cross}(D,E), \mathit{pop}(C,D,E)\big),$$

where

$$\begin{aligned}
\mathit{gen}(D,E) &= 1 + \max\big\{\textsc{gen}(D), \textsc{gen}(E)\big\},\\
\mathit{cross}(D,E) &= 1 + \big|\big\{B \in \textsc{anc}(D) \cup \textsc{anc}(E) \mid B \notin P\big\}\big|,\\
\mathit{pop}(C,D,E) &= N(C,D,E,\gamma)\\
&\qquad + \sum_{B \in \textsc{anc}(D) \cup \textsc{anc}(E)} \begin{cases} 1, & \text{if } B \in P,\\ N(B, p_1(B), p_2(B), \gamma), & \text{if } B \notin P. \end{cases}
\end{aligned}$$

From now on, we abbreviate $\mathit{cost}(\mathit{gen}(D,E), \mathit{cross}(D,E), \mathit{pop}(C,D,E))$ as $\mathit{cost}(C,D,E)$. Similarly to the previous subsection, we determine the number of crossings and the population size by taking the ancestors of C into account. Since we only store the latest generation, this time we can use $\textsc{anc}(D)$ and $\textsc{anc}(E)$ directly. We put genotype C in $\mathcal{L}_{\text{new}}$ only if

1. its cost is less than $A(C^*)$, i.e. the cost of the ideotype, and
2. its cost is less than $A(C)$, and
3. the population size needed to obtain C out of D and E is at most $N_{\max}$.

We mentioned earlier that the final step in an iteration consists of updating $\textsc{anc}(C)$ and $\textsc{gen}(C)$ for any genotype $C \in (\mathcal{L}_{\text{cur}} \cup \mathcal{L}_{\text{old}})$. We update these attributes recursively. In order to reduce the amount of work, we maintain a flag that signals whether the attributes of a genotype have already been updated. If during the recursion we encounter a genotype that has already been updated, we abort the recursion. In this way a genotype is considered only once.

We still have to show the soundness of this heuristic. In other words we want to show that no cycles are introduced in the crossing schedules defined in the table A. For that purpose we introduce the following invariant.

Invariant 5.2 *There are no genotypes C and D for which it holds that $C \in \text{ANC}(D)$ and $D \in \text{ANC}(C)$.*

Proof. We prove the invariant by induction on the number of iterations. Initially, when $i = 1$, the invariant holds, as we then have $\text{ANC}(C) = \emptyset$ for all genotypes C.

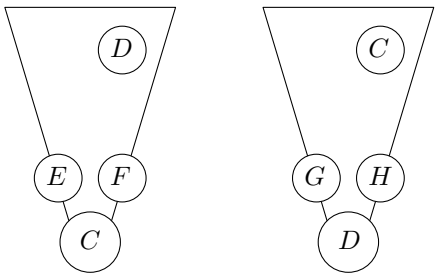


Figure 5.2: Only way to end up with a cycle.

As for the step, assume for a contradiction that at the end of iteration $i > 1$ there are two genotypes C and D such that $C \in \text{ANC}(D)$ and $D \in \text{ANC}(C)$. Let E and F be the parents of C, whereas G and H are the parents of D (cf. Figure 5.2). By the induction hypothesis, we have that C or D must be in $\mathcal{L}_{\text{new}}$. Now let's look at the moment just before $\mathcal{L}_{\text{new}}$ has been committed into A. The following two cases can be distinguished.

1. Both C and D are in $\mathcal{L}_{\text{new}}$.

   We have that $cost(D, G, H) < A[D]$ and $cost(C, E, F) < A[C]$. We also have that $A[C] < cost(D, G, H)$, as $C \in (\text{ANC}(G) \cup \text{ANC}(H) \cup \{G\} \cup \{H\})$. Thus we can conclude that $cost(C, E, F) < cost(D, G, H)$. Conversely, we have that $A[D] < cost(C, E, F)$, as $D \in (\text{ANC}(E) \cup \text{ANC}(F) \cup \{E\} \cup \{F\})$. Now we can conclude that $cost(D, G, H) < cost(C, E, F)$. Contradiction.

2. Only one of C and D is in $\mathcal{L}_{\text{new}}$.

   Assume that $(D, G, H) \in \mathcal{L}_{\text{new}}$. This means that $cost(D, G, H) < A[D]$. Since $D \in \text{ANC}(C)$, we have that $A[D] < A[C]$. We also have that $A[C] < cost(D, G, H)$, as $C \in (\text{ANC}(G) \cup \text{ANC}(H) \cup \{G\} \cup \{H\})$. Combining these two inequalities yields $A[D] < cost(D, G, H)$. Contradiction.

So the step holds and the lemma follows. □

In every iteration only the k lowest ranking genotypes are retained in $\mathcal{L}_{\text{new}}$. In order to do this, we need to describe what the order of the genotypes is. For that purpose, we introduce the following functions that assign a cost to a genotype C given a target chromosome C^*_x.

(f_1) The first function assigns a cost to C by simply counting the number of correct loci, i.e.

$$f_1(C, C^*_x) = \left|\left\{i \mid 0 \leq i < m \text{ and } (C_{0i} = C^*_{xi} \text{ or } C_{0i} = C^*_{xi})\right\}\right|.$$

(f_2) In the second function, we determine which chromosome of C has the largest number of correct alleles and assign the cost to be that number, i.e.

$$f_2(C, C_x^*) = \max\{g_2(C_0, C_x^*), g_2(C_1, C_x^*)\},$$

where

$$g_2(B_x, C_x^*) = \big|\{i \mid 0 \leq i < m \text{ and } B_{xi} = C_{xi}^*\}\big|.$$

(f_3) As for the third function, we obtain a chromosome D_x^* by considering every locus i where $0 \leq i < m$. We set D_{xi}^* to C_{0i} if i is a homozygous locus in C not containing the target allele C_{xi}^*. Otherwise we set D_{xi}^* to C_{xi}^*. So D_{xi}^* contains all the alleles that C_x^* and C have in common.

The cost of C is obtained by translating $P(C \to D_x^*)$ —the probability of obtaining D_x^* out of C (see Definition 2.2)— to a population size. An additional penalty term is used that penalizes the number of incorrect loci in C. The coefficient of this penalty term is the maximum of the cost of a generation and the cost of a crossing, i.e.

$$f_3(C, C_x^*) = \left\lceil \frac{\ln(1-\gamma)}{\ln(1 - P(C \to D_x^*))} \right\rceil + \max\big\{cost(1,0,0), cost(0,1,0)\big\} \cdot (m - f_1(C, C_x^*)).$$

(f_4) For the fourth function, we define the cost to be the cardinality of the largest subset of consecutive correct alleles resulting after one crossover. There are $m-1$ crossover points. Each crossover between loci i and $i+1$ can result in two chromosomes; $g_4(C, i)$ returns these two chromosomes. The largest subset of consecutive alleles C_x and C_x^* have in common is returned by $h_4(C_x, C_x^*)$. We define $f_4(C, C_x^*)$ as follows.

$$f_4(C, C_x^*) = \max_{0 \leq i < m-1} \big\{ \max_{B_x \in g_4(C,i)} \big\{h_4(B_x, C_x^*)\big\}\big\}.$$

(f_5) Contrary to the previous functions, this final function results in a pair of natural numbers, i.e.

$$f_5(C) = (a, b),$$

where

$$a = \min\left\{\sum_{i=0}^{m-1} 2^i \cdot C_{0i}, \sum_{i=0}^{m-1} 2^i \cdot C_{1i}\right\},$$
$$b = \max\left\{\sum_{i=0}^{m-1} 2^i \cdot C_{0i}, \sum_{i=0}^{m-1} 2^i \cdot C_{1i}\right\}.$$

In other words the first element of the pair is less than or equal to the second element. The function that we defined here induces a *well-ordering* on the set of possible chromosomes.

In Figure 5.3 an example in which the functions are used is given. Using these functions, we define the following *strict weak orderings*. Let C_x^* be the target chromosome that is considered.

($\prec_1$) This ordering makes use of $f_1(\cdot,\cdot)$ and in case of ties, it falls back to $\prec_2$.

$D \prec_1 E$ iff

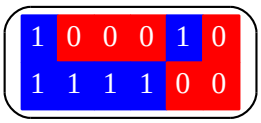
$$C^* = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$RM = \begin{pmatrix} 0 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0 \end{pmatrix}$$

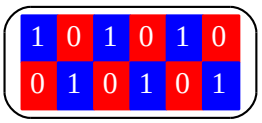
$$\gamma = 0.95$$

$$cost(g, c, p) = 100g + 150c + p$$

(a) Problem instance.



(b) Genotype D.



(c) Genotype E.

Cost of D			Cost of E			Comparison
$f_1(D, C_0^*)$	$=$	5	$f_1(E, C_0^*)$	$=$	6	$E \prec_1 D$
$f_2(D, C_0^*)$	$=$	4	$f_2(E, C_0^*)$	$=$	3	$D \prec_2 E$
$f_3(D, C_0^*)$	$=$	$\frac{\log(1-0.95)}{\log(1-0.5\cdot(1-0.1)^2\cdot 0.1)} + 150$	$f_3(E, C_0^*)$	$=$	$\frac{\log(1-0.95)}{\log(1-0.5\cdot 0.1^5)}$	$D \prec_3 E$
	$=$	223		$=$	599145	
$f_4(D, C_0^*)$	$=$	5	$f_4(E, C_0^*)$	$=$	2	$D \prec_4 E$
$f_5(D)$	$=$	$(34, 60)$	$f_5(E)$	$=$	$(21, 42)$	$E \prec_5 D$

(d) Genotype costs and comparisons.

Figure 5.3: Example of how the defined functions assign a cost to genotype.

- $f_1(D, C_x^*) > f_1(E, C_x^*)$, or
- $f_1(D, C_x^*) = f_1(E, C_x^*)$ and $D \prec_2 E$.

($\prec_2$) Now we make use of $f_2(\cdot, \cdot)$ and fall back to $\prec_4$ in case of ties.

$D \prec_2 E$ iff

- $f_2(D, C_x^*) > f_2(E, C_x^*)$, or
- $f_2(D, C_x^*) = f_2(E, C_x^*)$ and $D \prec_4 E$.

($\prec_3$) This time we use $f_3(\cdot, \cdot)$; we break ties using $\prec_5$.

$D \prec_3 E$ iff

- $f_3(D, C_x^*) < f_3(E, C_x^*)$, or
- $f_3(D, C_x^*) = f_3(E, C_x^*)$ and $D \prec_5 E$.

($\prec_4$) We use $f_4(\cdot, \cdot)$ for this ordering; again ties are broken using $\prec_5$.

$D \prec_4 E$ iff

- $f_4(D, C_x^*) > f_4(E, C_x^*)$, or
- $f_4(D, C_x^*) = f_4(E, C_x^*)$ and $D \prec_5 E$.

($\prec_5$) We now use $f_5(\cdot)$. The first element of the pair returned by $f_5(B)$ can be obtained using $first(f_5(B))$, whereas the second element of that pair is $second(f_5(B))$.

$D \prec_5 E$ iff

- $\mathit{first}(f_5(D)) < \mathit{first}(f_5(E))$, or
- $\mathit{first}(f_5(D)) = \mathit{first}(f_5(E))$ and $\mathit{second}(f_5(D)) < \mathit{second}(f_5(E))$.

In Figure 5.3d an example on the application of the strict weak orderings is given. If the ideotype C^* is heterozygous, i.e. $C_0^* \neq C_1^*$, we select $k/2$ genotypes according to C_0^* and $k/2$ genotypes according to C_1^*. In case C^* is homozygous, we simply select k genotypes according to either chromosome of C^*.

Now let's look at the running time and the space consumption. In order to do this, we need to bound the number of ancestors that a genotype can have. Contrary to the previous method, this time the iteration number does not correspond to the maximal generation present in A. The maximal generation at the end of iteration i is denoted by $d(i)$. We have that $d(i)$ is monotonously increasing in i. For the moment, we do not give a bound on $d(i)$.

The number of ancestors for any genotype in iteration i is $\mathcal{O}(2^{d(i)})$. Computing the cost of a genotype C resulting from crossing genotypes D and E at iteration i requires $\mathcal{O}(m+d(i)2^{d(i)})$ time. The first term stems from the computation of $N(C, D, E, \gamma)$, whereas the second term is because of the sorting that is required to compute the ancestor set. Since the number of genotypes that are crossed is $\mathcal{O}(2^{2m})$, we have that the total time needed for this procedure is $\mathcal{O}((d(i)2^{d(i)} + m)2^{2m})$.

Recall that initially $\mathcal{L}_{\text{cur}} = P$ and $\mathcal{L}_{\text{old}} = \emptyset$. Also recall that k is the number of genotypes that are retained from $\mathcal{L}_{\text{new}}$. Assuming that $n = \mathcal{O}(k)$ (where $n = |P|$), we have that at iteration i the cardinality of $\mathcal{L}_{\text{old}}$ is $\mathcal{O}(ik)$. Since at the end of every iteration $\mathcal{L}_{\text{cur}}$ takes the contents of $\mathcal{L}_{\text{new}}$ and $n = \mathcal{O}(k)$, we have that $|\mathcal{L}_{\text{cur}}| = \mathcal{O}(k)$.

Let's look at the total time needed updating the table A at the end of an iteration i. We can see that every genotype in $\mathcal{L}_{\text{cur}} \cup \mathcal{L}_{\text{old}}$ is updated. The cardinality of $\mathcal{L}_{\text{cur}} \cup \mathcal{L}_{\text{old}}$ at iteration i is $\mathcal{O}(ik)$. Only when a genotype has not been considered yet, we do some actual work. The work that we do there is dominated by the set union needed for computing the new ancestor set. For computing the set union we require a sorting step, which takes $\mathcal{O}(d(i)2^{d(i)})$ time in iteration i. So the time required for updating A in iteration i is $\mathcal{O}(ikd(i)2^{d(i)})$.

The cardinality of $\mathcal{L}_{\text{cur}} \bar{\times} \mathcal{L}_{\text{cur}}$ is $k(k+1)/2 = \mathcal{O}(k^2)$. On the other hand, the cardinality of $\mathcal{L}_{\text{cur}} \times \mathcal{L}_{\text{old}}$ is $\mathcal{O}(ik^2)$. So the time required for crossing the genotypes in $\mathcal{L}_{\text{cur}} \times \mathcal{L}_{\text{old}}$ is more than the time required for $\mathcal{L}_{\text{cur}} \bar{\times} \mathcal{L}_{\text{cur}}$. The time required is thus $\mathcal{O}((d(i)2^{d(i)} + m)ik^2 2^{2m})$, which dominates the time for updating A.

The only time consuming step we did not look at is the computation of $\mathcal{L}_{\text{new}}$. The cardinality of $\mathcal{L}_{\text{new}}$ is $\mathcal{O}(2^{2m})$. If we want to obtain the k lowest ranking genotypes in $\mathcal{L}_{\text{new}}$, we can employ k times a linear-time selection algorithm [2]. The selection algorithm is comparison based. In our case resolving a comparison between two genotypes requires linear time (even the evaluation of $f_4(\cdot,\cdot)$ can be done in linear time). So obtaining the k lowest ranking genotypes requires $\mathcal{O}(km2^{2m})$ time.

We can now conclude that the time required for g iterations is

$$\begin{aligned}\sum_{i=1}^{g} \mathcal{O}((d(i)2^{d(i)} + m)ik^2 2^{2m} &= \mathcal{O}(k^2 2^{2m}) \sum_{i=1}^{g} \mathcal{O}(id(i)2^{d(i)}) + \mathcal{O}(mk^2 2^{2m}) \sum_{i=1}^{g} \mathcal{O}(i) \\ &= \mathcal{O}(g^2 d(g) 2^{d(g)} k^2 2^{2m}) + \mathcal{O}(g^2 mk^2 2^{2m}) \\ &= \mathcal{O}((d(g)2^{d(g)} + m)g^2 k^2 2^{2m}).\end{aligned}$$

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \Longrightarrow \begin{array}{l} \{ \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}, \\ \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix} \} \end{array}$$

Figure 5.4: The number of gametes is 2^l, where l is the number of heterozygous loci. The first two gametes result from two crossovers, the four after those require one crossover and the two final ones require no crossovers.

Note that in the derivation we could not bound $\sum_{i=1}^{g} \mathcal{O}(id(i)2^{d(i)})$ tightly by an infinitely decreasing geometric series, as $d(i)$ is not strictly increasing in i.

After g iterations the number of genotypes in A is $\mathcal{O}(gk)$. The storage of A is dominated by the storage of the ancestors for every genotype and is thus $\mathcal{O}(gk2^{d(g)})$. We mentioned earlier that $\mathcal{L}_{\text{new}}$ has cardinality $\mathcal{O}(2^{2m})$. Therefore the space consumption is $\mathcal{O}(gk2^{d(g)} + 2^{2m})$. The results of this subsection are summarized in the following theorem.

Theorem 5.3 *Restricting the number of genotypes results in a heuristic requiring* $\mathcal{O}((d(g)2^{d(g)} + m)g^2k^2 2^{2m})$ *time and consuming* $\mathcal{O}(gk2^{d(g)} + 2^{2m})$ *space.*

Since we have to pay for every additional generation and the crossing it introduces, it turns out that in practice $d(i) = \mathcal{O}(i)$ with a very low constant hidden in the big O. We think that by employing an accounting scheme the following conjecture can be proven true.

Conjecture 5.4 *At the end of iteration* i, $d(i) = \mathcal{O}(i)$.

5.1.3 Restricting gametes

Our goal is to obtain a heuristic whose running time is polynomial in n and m—where n is the number of parents and m is the number of loci. By limiting the number of genotypes that continue to a next iteration, we almost succeeded in achieving this goal. The only thing that makes the running time exponential in m is the procedure Cross. In this subsection we will show how this procedure can be made to run in polynomial time.

Let's take a closer look at a genotype C. The number of loci in C is m. We mentioned earlier that the number of gametes that C can give rise to is 2^l, where l is the number of heterozygous loci in C (cf. Figure 5.4). We can classify the gametes that result from C, by taking the number of required crossovers into account. For instance, to obtain $\begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$ from $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$ we require two crossovers. Between any two consecutive loci a crossover can occur. So the number of crossover points is $m-1$. In our example the crossover points that are used can be identified by looking at the colors: we have a crossover between the first and second locus and a crossover between the third and fourth locus. Using the same crossover points we can also obtain $\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$.

Now let's generalize the previous example. Let D_0 be a gamete that can result from a genotype C. The gamete D_0 is obtained by a sequence of crossovers in C. A crossover occurs at one crossover point. So if D_0 is obtained by b crossovers in C then b crossover points are used. We now want to bound the number of gametes obtained by b crossovers. To have this many crossovers, we need to pick b crossover points. There are $\binom{m-1}{b}$ ways to do this. Every picked

subset of crossover points can result in at most two different gametes. Thus the number of gametes obtained by b crossovers is bounded by

$$2 \cdot \binom{m-1}{b} = \mathcal{O}(m^b).$$

The idea now is to introduce a constant c such that only gametes that can be obtained with at most c crossovers are allowed. The number of such gametes is

$$\sum_{i=1}^{c} \mathcal{O}(m^i) = \mathcal{O}(m^c).$$

If we restrict the gametes that are considered during a crossing using c, the number of considered *genotypes* becomes $O(m^{2c})$. Now performing a crossing at iteration i takes $\mathcal{O}((d(i)2^{d(i)} + m)m^{2c})$ time as opposed to $\mathcal{O}((d(i)2^{d(i)} + m)2^{2m})$ time.

The running time of crossing the genotypes in $\mathcal{L}_{\text{cur}} \times \mathcal{L}_{\text{old}}$ in iteration i is $\mathcal{O}((d(i)2^{d(i)} + m)ik^2m^{2c})$. Now let's look at the time required for determining the k genotypes that are to remain in $\mathcal{L}_{\text{new}}$. Since we cross $\mathcal{O}(ik^2)$ times, we have that the number of genotypes that are added in iteration i to $\mathcal{L}_{\text{new}}$ is $\mathcal{O}(ik^2m^{2c})$. By again employing a linear time selection algorithm, we can select the k genotypes with ranks $1 \ldots k$ in $\mathcal{O}(imk^3m^{2c})$ time. The additional factor m in the running time is due to the linear time comparisons that are needed. All together we have

$$\begin{aligned}\sum_{i=1}^{g} \mathcal{O}((d(i)2^{d(i)} + m)ik^2m^{2c} + imk^3m^{2c}) &= \mathcal{O}((d(g)2^{d(g)} + m)g^2k^2m^{2c} + g^2mk^3m^{2c}) \\ &= \mathcal{O}((d(g)2^{d(g)} + mk)g^2k^2m^{2c}).\end{aligned}$$

Now let's look at the space consumption. We showed earlier that the cardinality of $\mathcal{L}_{\text{new}}$ at iteration i is $\mathcal{O}(ik^2m^{2c})$. So at iteration g the cardinality of $\mathcal{L}_{\text{new}}$ is $\mathcal{O}(gk^2m^{2c})$. We also have that the number of entries in A at the end of the final iteration is $\mathcal{O}(gk)$. For each entry we store the ancestors; the number of ancestors is $\mathcal{O}(2^{d(g)})$. Therefore the total space consumption is $\mathcal{O}(gk^2m^{2c} + gk2^{d(g)})$. In the following theorem the result that we have obtained is given.

Theorem 5.5 *Restricting the number of genotypes and the number of gametes results in a heuristic requiring $\mathcal{O}((d(g)2^{d(g)} + mk)g^2k^2m^{2c})$ time and consuming $\mathcal{O}(gk^2m^{2c} + gk2^{d(g)})$ space.*

The running time is now polynomial in both n and m. So we have achieved our goal of having a polynomial-time heuristic. The only thing that we did not describe yet is what the justification of restricting the number of crossovers is. Every crossover needed to obtain D_0 out of genotype C, corresponds to a multiplication by the recombination frequency of the loci involved in the computation of $P(C \to D_0)$. The more crossovers we need for obtaining a gamete, the lower the probability for obtaining that gamete will become and consequently the population size needed will also increase. It turns out that there is an upper bound on c by taking $N_{\max}$ into account. Let r be the maximal element in RM. By Definition 2.1, we

have that $r \leq 0.5$. In the worst case all c crossovers have a recombination frequency of r and we would then have the following inequality relating r with c and $N_{\max}$.

$$\frac{\ln(1-\gamma)}{\ln(1-r^c)} \leq N_{\max}.$$

Isolating c results in

$$c \leq \frac{\ln(1-(1-\gamma)^{\frac{1}{N_{\max}}})}{\ln(r)}.$$

So choosing a value for $N_{\max}$ corresponds to limiting the number of crossovers. The bound that we have just given is not a tight one in practice. For instance, in the derivation of the bound we did not take into account that we have to pay for not having a crossover as well. Furthermore, there is an additional multiplication by 0.5 that we did not include.

5.1.4 Additional improvements

In this final subsection we will describe three additional improvements on the heuristic that we developed in the previous subsections.

Further gamete restriction. We start by describing a method for further restricting the number of gametes that are considered in the procedure Cross. This method can be applied on top of the method described in the previous subsection. In Subsection 5.1.2 we defined $h_4(C_x, C_x^*)$ to be the cardinality of the largest subset of consecutive alleles that C_x and C_x^* have in common. We determine the maximal number of consecutive correct alleles either chromosome of a genotype C has as follows.

$$a = \max\left\{h_4(C_0, C_0^*), h_4(C_1, C_0^*), h_4(C_0, C_1^*), h_4(C_1, C_1^*)\right\}.$$

Let B_0 be a gamete that can result from C. The idea is to only consider B_0 in the procedure Cross if it is at least as good as C. We do this by defining b as

$$b = \max\left\{h_4(B_0, C_0^*), h_4(B_0, C_1^*)\right\}.$$

So b denotes the cardinality of the largest subset of consecutive correct alleles in B_0. Gamete B_0 is only as good as C if $b \geq a$. If this is not the case, we do not consider B_0 in Cross. Note that this method does not improve the asymptotic running time. A simple counterexample for this is obtained by defining the chromosomes that comprise C to contain 0 and 1 alleles alternatingly in such a way that $C_0 \neq C_1$:

$$C = \begin{pmatrix} 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 1 \end{pmatrix}.$$

In case $C^* = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}$, we have that the cardinality of the largest subset of consecutive correct alleles is 1. So the only gamete resulting from C that is disallowed is $\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \end{pmatrix}$. Therefore the number of considered gametes does not change asymptotically.

Homozygous genotypes. The second improvement that we describe is based on the observation that a homozygous genotype can be obtained optimally via a selfing. An even stronger claim can be proven if the population size is also taken into account in the cost function.

Lemma 5.6 *In case we pay for the population size, i.e. cost*$(0, 0, 1) > 0$*, a homozygous genotype can* only *be obtained optimally via a selfing.*

Proof. We prove this by contradiction. Let C be a homozygous genotype obtained optimally but not via a selfing. Let D and E be the two parents of C. Since C is homozygous, that is $C_0 = C_1$, we have that $P(D \to C_0) > 0$ and $P(E \to C_0) > 0$. In other words, both D and E can give rise to C_0.
Now let's look at what happens when we self either D or E. The number of generations in the worst case would be the same as it initially was, as initially we had $\textsc{gen}(C) = 1 + \max\{\textsc{gen}(D), \textsc{gen}(E)\}$. Similarly, the number of crossings would not change for the worse, as initially the number of crossings of C, denoted by $\textsc{crs}(C)$, is bounded as follows.

$$\textsc{crs}(C) \geq 1 + \textsc{crs}(D) \text{ and } \textsc{crs}(C) \geq 1 + \textsc{crs}(E).$$

Let's look at the population size. We claim that if we self the genotype that has the highest probability of giving rise to C_0, we can obtain C more cheaply. Since C is homozygous, we have by Lemma 2.3 that

$$P(D, E \to C) = P(D \to C_0) \cdot P(E \to C_0).$$

We assume without loss of generality that $P(E \to C_0) \leq P(D \to C_0)$. Therefore we have

$$P(D, E \to C) \leq P(D \to C_0)^2.$$

So the population size needed for obtaining C by selfing D is at most the population size that we needed to obtain C out of D and E. When we obtain C by selfing D, we have no use for E. So we do not need to pay for the population size needed to obtain E out of its parents. This population size is at least 1. Note that even if E has no parents, i.e. $E \in P$, the population size of E would still be 1 (see Equation 2.2). So by selfing D, the *total* population size needed is at least one less than initially was the case. Earlier we argued that the number of crossings and the number of generations do not change for the worse when we self D. Hence, we can obtain C more optimally by selfing D. □

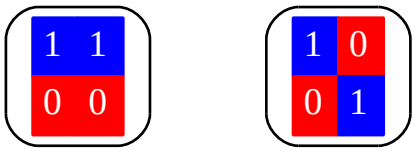
With exactly the same arguments that were used to show that the number of generations and the number of crossings do not change for the worse in case of selfing the following lemma can be proven true.

Lemma 5.7 *In case we do* not *pay for the population size, i.e. cost*$(0, 0, 1) = 0$*, a homozygous genotype can be obtained optimally via a selfing.*

The two lemmas combined lead to the following corollary.

Corollary 5.8 *A homozygous genotype can be obtained optimally via a selfing.*

We make use of this corollary by disallowing a homozygous genotype to result out of two different genotypes in the procedure Cross. While this has no effect asymptotically, it does save a few unnecessary computations.



(a) Coupling phase. (b) Repulsion phase.

Figure 5.5: Coupling and repulsion phase of alleles of linked loci.

$$P = \left\{ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \right\}$$
$$C^* = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

		locus index			
		0	1	2	3
locus index	0	-	coupled	unlinked	unlinked
	1	coupled	-	repulsed	unlinked
	2	unlinked	repulsed	-	unlinked
	3	unlinked	unlinked	unlinked	-

Figure 5.6: Example of loci pair classification.

Phase synchronization. We call the final improvement that we consider *phase synchronization*. In genetics, two alleles located on loci that are linked can either be in repulsion phase or in coupling phase [6]. We say that two alleles are in *coupling phase* if they are present on the same homologous chromosome (cf. Figure 5.5a). Otherwise we say that they are in *repulsion phase* (cf. Figure 5.5b). We classify all $\binom{m}{2}$ loci pairs in the following way.

- Loci pair (i, j) is *unlinked* present in P with respect to C^*_x if there is no genotype $D \in P$ in which the two alleles C^*_{xi} and C^*_{xj} are present, i.e.

$$(C^*_{xi} \neq C_{0i} \text{ and } C^*_{xi} \neq C_{1i}) \text{ or } (C^*_{xj} \neq C_{0j} \text{ and } C^*_{xj} \neq C_{1j}).$$

- Loci pair (i, j) is *coupled* present in P with respect to C^*_x if there is a genotype $C \in P$ in which the two alleles C^*_{xi} and C^*_{xj} are present on one chromosome, i.e.

$$(C^*_{xi} = C_{0i} \text{ and } C^*_{xj} = C_{0j}) \text{ or } (C^*_{xi} = C_{1i} \text{ and } C^*_{xj} = C_{1j}).$$

- Loci pair (i, j) is *repulsed* present in P with respect to C^*_x if (i, j) is not unlinked nor coupled with respect to C^*_x.

In Figure 5.6 an example of the just described classification is given. In order to obtain the ideotype, we must first attain genotypes in which the alleles of unlinked loci pairs are in repulsion phase. Subsequently, these repulsed alleles must become in coupling phase. The latter stage is handled perfectly fine by the strict weak orderings introduced in Subsection 5.1.2. The first stage, however, may be a bit more problematic. For instance, it can be the case that only two parents provide the target alleles of an unlinked loci pair. A genotype that results from these two parents and in which the alleles of the unlinked loci pair have become in repulsion phase may be unfavorable when compared to other genotypes. Nonetheless, the genotype is crucial for attaining the ideotype.

To properly handle crucial genotypes, we introduce the method of phase synchronization. In this method we first identify all unlinked loci pairs with respect to C^*_0 and C^*_1. At the end of

every iteration of the heuristic, we make sure that for every unlinked loci pair l genotypes in which the corresponding alleles are either in repulsion or in coupling phase have been added to $\mathcal{L}_{\text{cur}}$. Note that we cannot always select exactly l genotypes per unlinked loci pair, as it can be the case that for a particular unlinked loci pair less than l genotypes with the desired property have been generated. The selection of the l genotypes is done by taking the ordering into account.

Let's determine the running time of this new method. The number of genotypes that can be added to $\mathcal{L}_{\text{cur}}$ due to phase synchronization is $\mathcal{O}(lm^2)$. So the cardinality of $\mathcal{L}_{\text{cur}}$ is $\mathcal{O}(k+lm^2)$ and consequently the cardinality of $\mathcal{L}_{\text{old}}$ is $\mathcal{O}(ik + ilm^2)$. Therefore the number of pairs in $\mathcal{L}_{\text{cur}} \times \mathcal{L}_{\text{old}}$ at iteration i is $\mathcal{O}((k + lm^2)(ik + ilm^2)) = \mathcal{O}(i(k^2 + klm^2 + l^2m^4))$. To simplify the computation, we bound the cardinality of $\mathcal{L}_{\text{cur}} \times \mathcal{L}_{\text{old}}$ more loosely as $\mathcal{O}(ik^2l^2m^4)$. Recall that crossing two genotypes takes $\mathcal{O}((d(i)2^{d(i)} + m)m^{2c})$ time. So crossing all genotype pairs in $\mathcal{L}_{\text{cur}} \times \mathcal{L}_{\text{old}}$ takes $\mathcal{O}((d(i)2^{d(i)} + m)ik^2l^2m^{2c+4})$ time.

This time we cannot avoid sorting $\mathcal{L}_{\text{new}}$, as we do not know the rank of the l genotypes in advance. The cardinality of $\mathcal{L}_{\text{new}}$ at iteration i is $\mathcal{O}(ik^2l^2m^{2c+4})$. Assuming that $i, l, m \leq k$, the time required for sorting $\mathcal{L}_{\text{new}}$ is

$$\mathcal{O}(ik^2l^2m^{2c+4} \log(ik^2l^2m^{2c+4})) = \mathcal{O}(ik^2l^2m^{2c+4}c \log(iklm)) = \mathcal{O}(ik^2l^2m^{2c+4}c \log k).$$

The time required for one iteration of the heuristic is thus $\mathcal{O}((d(i)2^{d(i)}+m+c \log k)ik^2l^2m^{2c+4})$. So the total time required is

$$\begin{aligned}
&\sum_{i=1}^{g} \mathcal{O}((d(i)2^{d(i)} + m + c \log k)ik^2l^2m^{2c+4}) \\
&= \mathcal{O}(k^2l^2m^{2c+4}) \left[\sum_{i=1}^{g} \mathcal{O}(id(i)2^{d(i)}) + \sum_{i=1}^{g} \mathcal{O}(im + ic \log k)\right] \\
&= \mathcal{O}((d(g)2^{d(g)} + m + c \log k)g^2k^2l^2m^{2c+4}).
\end{aligned}$$

Finally we look at the space consumption. We have that the number of genotypes in A at the end of the final iteration is $\mathcal{O}(gk + glm^2)$. We simplify this to $\mathcal{O}(gkm^2)$ by assuming that $l \leq k$. For every genotype we have to store its ancestors; the number of ancestors for a genotype is $\mathcal{O}(2^{d(g)})$. We also have to look at the space consumption of $\mathcal{L}_{\text{new}}$. In generation g, the cardinality of $\mathcal{L}_{\text{new}}$ is $\mathcal{O}(gk^2l^2m^{2c+4})$. Therefore the total space consumption is $\mathcal{O}((m^2 2^{d(g)} + kl^2m^{2c+4})gk)$.

Theorem 5.9 *In addition to restricting the number of genotypes and gametes, applying phase synchronization results in a heuristic requiring* $\mathcal{O}((d(g)2^{d(g)} + m + c \log k)g^2k^2l^2m^{2c+4})$ *time and consuming* $\mathcal{O}((m^2 2^{d(g)} + kl^2m^{2c+4})gk)$ *space.*

As a final remark, we point out that a lower bound on $N_{\max}$ can be determined by looking at the recombination frequencies of unlinked and repulsed loci pairs. Multiplying the minimal one among these by 0.5 and subsequently translating the resulting value to a population size results in the lower bound. No crossing schedule resulting in the ideotype exists, if $N_{\max}$ is set to a value lower than the lower bound.

5.2 Randomized algorithm

In this section we describe a randomized algorithm for solving the problem. The algorithm we present is based on the method used for generating the initial population in the genetic algorithm. Similarly to Section 4.2, we restrict a node in the schedule to only give rise to one gamete. The notation is kept the same: the gamete given rise to by a node v is denoted by $\mathit{gamete}(v)$ and the genotype of that node is denoted by $\mathit{genotype}(v)$. Contrary to the discussion of the genetic algorithm, we do not restrict a crossing schedule to be a tree. Instead, we view a crossing schedule as a DAG with one target node. The source nodes of the DAG correspond to original parents in P, whereas non-source nodes represent crossings between two parents. The genotype of a non-source node v whose parents are u and w is $(\mathit{gamete}(u), \mathit{gamete}(w))$, whereas the genotype of a source node is simply the genotype of its corresponding parent.

Throughout this discussion we assume that the ideotype is homozygous, i.e. $C_0^* = C_1^*$. The idea is to define a recursive algorithm that takes as input a subset $P' \subseteq P$ and results in a pair $(G_{P'}, v^*)$ where

- $G_{P'}$ is a DAG whose source nodes are labeled with the genotypes in P', and
- v^* is a node in $G_{P'}$ such that $\mathit{genotype}(v^*)$ is homozygous.

The rationale for returning v^* is to facilitate expensive crossovers: due to the homozygosity of v^* the probability of obtaining a genotype by crossing v^* and another node v is solely determined by v.

If in a recursive invocation the cardinality of P' is 1, we generate a source node v whose genotype C corresponds to the single parent in P'. The gamete of v is chosen uniformly at random to be either C_0 or C_1. If C is homozygous, we return the pair (G_v, v) where G_v is the graph consisting of only the node v. Otherwise if C is heterozygous, we self, with probability 0.5, node v and obtain a new node w whose genotype is $(\mathit{gamete}(v), \mathit{gamete}(v))$; subsequently the pair (G_{vw}, w) is returned where G_{vw} is the directed graph consisting of nodes v, w and a directed edge from v to w. If it was decided not to self v, $(G_v, \mathbf{nil})$ is returned.

In case $|P'| > 1$, we randomly partition P' in two subsets P'_1 and P'_2. Subsequently, we recurse on both subsets and obtain two pairs (G_1, v_1^*) and (G_2, v_2^*). The node v^* is defined as

$$v^* = \begin{cases} v_1^*, & \text{if } P((\mathit{gamete}(v_1^*), C_0^*) \to C^*) > P((\mathit{gamete}(v_2^*), C_0^*) \to C^*), \\ v_2^*, & \text{otherwise.} \end{cases} \tag{5.1}$$

So v^* is the node among v_1^* and v_2^* that has the highest probability of participating in the final selfing resulting in C^*. The target nodes of G_1 and G_2 are denoted by t_1 and t_2, respectively. We generate a new node v by merging G_1 and G_2 such that $\mathit{genotype}(v) = (\mathit{gamete}(t_1), \mathit{gamete}(t_2))$. The gamete produced by v is chosen in such a way that the number of alleles it has in common with C_0^* is maximal. If $v^* \neq \mathbf{nil}$ then we backcross, with probability 0.5, v with v^* and obtain a node w whose genotype is $(\mathit{gamete}(v), \mathit{gamete}(v^*))$. Again the gamete given rise to by w is chosen such that it mostly resembles C_0^*. After the optional backcross an optional selfing step is performed with a probability 0.2; this selfing results in a node x such that

$$\mathit{genotype}(x) = \begin{cases} (\mathit{gamete}(w), \mathit{gamete}(w)), & \text{if a backcross was performed,} \\ (\mathit{gamete}(v), \mathit{gamete}(v)), & \text{otherwise.} \end{cases}$$

So invoking the recursive algorithm on P' results in the following.

1. In case of *no backcross and no selfing*, (G, v^*) is returned where G is the graph obtained by merging G_1 and G_2 extended with the node v and directed edges from t_1 to v and from t_2 to v.

2. In case of *a backcross and no selfing*, (G, v^*) is returned where G is the graph obtained by merging G_1 and G_2 extended with the nodes v and w, and directed edges from t_1 to v, from t_2 to v, from v to w and from v^* to w.

3. In case of *no backcross and a selfing*, $(G, f(v^*, x))$ is returned where $f(v^*, x)$ is defined as

$$f(v^*, x) = \begin{cases} v^*, & \text{if } P((gamete(v^*), C_0^*) \to C^*) > P((gamete(x), C_0^*) \to C^*), \\ x, & \text{otherwise,} \end{cases} \tag{5.2}$$

   and where G is the graph obtained by merging G_1 and G_2 extended with the nodes v and x, and directed edges from t_1 to v, from t_2 to v and from v to x.

4. In case of *a backcross and a selfing*, $(G, f(v^*, x))$ is returned where G is the graph obtained by merging G_1 and G_2 extended with the nodes v, w and x, and directed edges from t_1 to v, from t_2 to v, from v to w, from v^* to w and from w to x. Node $f(v^*, x)$ is defined as in Equation 5.2.

When determining $gamete(v)$ for any node v, we do not consider all gametes that can result out of D as was the case. Rather, we only allow gametes that are obtained with at most c crossovers (in a way similar to Subsection 5.1.3). In addition to this, we also disallow gametes whose probabilities of occurring are less than $p_{\min}$. The threshold $p_{\min}$ is the probability corresponding to $N_{\max}$, which can be shown to be equal to

$$p_{\min} = 1 - (1 - \gamma)^{\frac{1}{N_{\max}}}.$$

The procedure in which the previous is performed is denoted by Cross(P, c). The number of recursive invocations that are made when invoking Cross(P, c) is $2n - 1 = \mathcal{O}(n)$ (where $n = |P|$). In every recursive invocation, we need to randomly select a constant number (in fact at most two) of gametes under the restrictions of $N_{\max}$ and c. Recall that the number of gametes obtained by at most c crossovers is $\mathcal{O}(m^c)$. Picking a gamete with at most c crossovers and whose probability of occurring is at least $p_{\min}$ can be done in $\mathcal{O}(m^{c+1})$ time. So the total time required for computing Cross(P, c) is $\mathcal{O}(nm^{c+1})$.

The idea now is to invoke the procedure Cross multiple times and return the best crossing schedule among all invocations. Instead of invoking Cross every time with P, we use subsets $P' \subseteq P$ such that every allele in C^* is present in at least one parent in P'. To do this efficiently, we first determine the subset $P_{\min}$ containing the parents that provide alleles of C^* that no other parent in P provides. Subsequently, we extend $P_{\min}$ by randomly including other parents in P such that we obtain a subset P' that covers all target alleles. We invoke Cross l times using P'. The whole procedure is repeated k times.

The following theorem summarizes the result of this section.

Theorem 5.10 *If C^* is homozygous, a randomized algorithm that requires $\mathcal{O}(klnm^{c+1})$ time can be applied.*

Note that it can occur that in the procedure Cross a certain genotype is generated twice. That is why in our implementation we maintain a map mapping genotypes to nodes. Whenever we generate a genotype that is already present in the map we make use of the associated node. The final remark we make is about the homozygosity of C^*. By Corollary 5.8, we have that C^* requires a final selfing step. Since the probability of this final selfing step to occur is rather low, we perform it manually if it was not already performed.

Chapter 6

Experimental Evaluation

In this chapter, we evaluate the methods that have been introduced in the previous two chapters. We do this by comparing the methods in terms of computation time needed and the cost of a returned crossing schedule. In Section 6.1, we describe the experimental setup. Subsequently, the results are presented and discussed in Section 6.2.

6.1 Experimental setup

The methods that we want to evaluate are as follows.

1. Genetic algorithm (described in Section 4.2).

   In order to account for the stochasticity of the genetic algorithm, we run it ten times for 100 generations. The best crossing schedule among the ten repetitions is returned. We denote this method by `GA`.

2. Randomized algorithm (described in Section 5.2).

   The number of iterations of the randomized algorithm is chosen to be 100,000 (we generate 10,000 covers and every cover is used ten times). The constant c, which denotes the maximal number of crossovers, is set to 2. We denote this method by `RA`.

3. Dynamic programming heuristic (described in Section 5.1).

   Recall that we denote by k the number of genotypes that are retained per iteration. The dynamic programming heuristic in which $k = \infty$ is denoted by `DP`. On the other hand, the dynamic programming heuristic that makes use of $\prec_x$ as the strict weak ordering (cf. 5.1.2) and retains at most a constant number of genotypes per iteration is denoted by `DP`x`-`k (where $1 \leq x \leq 5$ and $k \geq 0$). If on top of the previous, phase synchronization is enabled, i.e. the number of genotypes selected due to phase synchronization l is greater than 0, we denote the dynamic programming heuristic by `DP`x`-`k`-`l. Finally, if further gamete restriction is applied then the method name is suffixed with a `'g'`. Similarly to `RA`, we set the constant c to 2.

We fix the number of loci to six. We choose the cost function such that the coefficients of the number of crossings and the number of generations are in the same order as the maximal population size (we describe later that $N_{\max}$ is set to 500):

$$cost(gen, cross, pop) = 100 \cdot gen + 100 \cdot cross + pop.$$

Furthermore we consider a homozygous ideotype. Using these constraints, we want to answer the following questions.

Q1. Does the randomized algorithm outperform the genetic algorithm?

Q2. Do the dynamic programming heuristics outperform the genetic algorithm?

Q3. What is the effect of varying the number of genotypes retained per iteration in the dynamic programming heuristics?

Q4. What is the effect of using different strict weak orderings in the dynamic programming heuristics?

Q5. What is the effect of applying phase synchronization in the dynamic programming heuristics?

Q6. What is the effect of further gamete restriction in the dynamic programming heuristics?

Q7. Does a heterozygous ideotype lead to different answers on questions Q2-Q6?

Q8. Does a different cost function lead to different answers on questions Q1 and Q2?

Q9. Does a different number of loci lead to different answers on questions Q1 and Q2?

In order to answer these question, we first have to generate problem instances. The input generator that we have developed for this purpose takes as arguments:

1. Number of loci, denoted by m.
2. Number of parents, denoted by n.
3. Ideotype, denoted by C^*.
4. Maximum number of correct alleles per chromosome, denoted by a.
5. Probability of having a homozygous parent, denoted by p_{hom}.
6. Maximum population size, denoted by $N_{\max}$.
7. Desired probability of success, denoted by γ.
8. Coefficients of the linear cost function.

The input generator generates n parents whose genotypes are comprised by m loci each. The chromosomes of the genotype of any generated parent are chosen such that they have each at most a alleles in common with both C_0^* and C_1^*. The genotype of a generated parent is chosen to be homozygous with probability p_{hom}. The recombination matrix RM is obtained by applying Haldane's mapping function (see Equation 1.2) on a randomly generated genetic map. In this map, the genetic distance between two consecutive loci is at least 1.5 cM and at most 30 cM. Note that the n parents are chosen in such a way that they can result in the ideotype, i.e. every allele in C^* is present in at least one parent.

Using the input generator, we generated five datasets. The attributes by which the datasets were generated are given in Table 6.1a. The methods that were run on every dataset are given in Table 6.1b. In Table 6.1c, we show for every dataset the number of iterations used for the dynamic programming heuristics. Since the method `DP` is computationally the most expensive, we only ran it on the datasets with six loci.

6.2 Results

All the methods have been implemented in C++ and are executed on a Intel Core 2 Duo E8400 3.0 GHz system with 4 GB RAM under Ubuntu 9.04 x86_64. For every run we collect the CPU time needed and the cost of the returned schedule. The costs cannot be used directly to compare different methods over different problem instances. Therefore we compute, per problem instance, the ratio between the cost returned by the method under consideration and the lowest cost among all methods ran on that problem instance. So for any problem instance the method that returned the cheapest crossing schedule has ratio 1.

In Figure 6.1 and Figure 6.2 the results are shown. Using these results we can answer the questions posed in the previous section. We answer questions Q1-Q6 by considering dataset 6loc-hom (note that this dataset adheres to the constraints specified earlier). Questions Q7 and Q8 are answered using datasets 6loc-het and 6loc-hom-pop, respectively. The last question Q9, is answered using datasets 10loc-hom and 15loc-hom.

Q1. Does the randomized algorithm outperform the genetic algorithm?

The mean cost ratio for the randomized algorithm is 1.263, whereas the genetic algorithm has a mean cost ratio of 3.530. So the randomized algorithm outperforms the genetic algorithm roughly three times in terms of the cost of a returned schedule.

The mean computation time for the randomized algorithm is 2.90 s, while for the genetic algorithm this is 96.29 s. Therefore, the randomized algorithm outperforms the genetic algorithm in terms of computation time as well. So based on the current data the answer on this question is yes.

Q2. Do the dynamic programming heuristics outperform the genetic algorithm?

In Figure 6.1a, we can see that all the dynamic programming heuristics have a lower mean cost ratio than the genetic algorithm. Except for the method `DP`, the mean computation time for the dynamic programming heuristics is also less than the mean computation time of the genetic algorithm.

	6loc-hom	**6loc-het**	**6loc-hom-pop**	**10loc-hom**	**15loc-hom**
m	6	6	6	10	15
n	$\{5, 10, 15, 20\}$	$\{5, 15\}$	$\{10, 20\}$	$\{10, 20\}$	$\{10, 20\}$
C^*	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$	all one alleles	all one alleles	all one alleles
a	$\{1, 2, 3, 4\}$	$\{3, 4\}$	$\{1, 2, 3, 4\}$	$\{1, 3, 5, 7\}$	$\{2, 5, 7, 10\}$
p_{hom}	$\{0.5, 0.9\}$	$\{0.5, 0.9\}$	0.9	0.9	0.9
$N_{\max}$	500	500	500	500	500
cost	$100 \cdot gen$ $+ 100 \cdot cross$ $+ pop$	$100 \cdot gen$ $+ 100 \cdot cross$ $+ pop$	pop	$100 \cdot gen$ $+ 100 \cdot cross$ $+ pop$	$100 \cdot gen$ $+ 100 \cdot cross$ $+ pop$
#repetitions	20	20	5	5	5
#instances	640	160	40	40	40

(a) Datasets.

	GA	RA	DP	DP1-100	DP1-150	DP2-100	DP2-150	DP3-100	DP3-150	DP4-10	DP4-70	DP1-150-5	DP2-150-5	DP3-150-5	DP4-70-5	DP1-150-5g	DP2-150-5g	DP3-150-5g	DP4-70-5g
6loc-hom	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
6loc-het	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
6loc-hom-pop	x	x														x	x	x	x
10loc-hom	x	x														x	x	x	x
15loc-hom	x	x														x	x	x	x

(b) Methods (abbreviations are explained in Section 6.1).

Method	g
6loc-hom	7
6loc-het	7
6loc-hom-pop	7
10loc-hom	9
15loc-hom	10

(c) Number of iterations of the dynamic programming heuristics.

Table 6.1: Datasets and methods.

In particular, method `DP4-70-5g` achieves a cost ratio of 1.004 and a mean computation time of 0.61 s. Based on the current observations, the answer on this question is thus yes.

Q3. What is the effect of varying the number of genotypes retained per iteration in the dynamic programming heuristics?

From the plots in Figure 6.1 we can see that increasing the number of genotypes retained results in a decrease of the cost ratio in all methods. While at the same time, the computation time needed increases. So increasing the number of genotypes retained enhances the performance in terms of the cost of the returned schedule at the expensive of more computation time.

Q4. What is the effect of using different strict weak orderings in the dynamic programming heuristics?

In the following table the results are summarized.

method	cost	time (s)
`DP1-150`	1.007	16.72
`DP2-150`	1.012	14.75
`DP3-150`	1.042	12.98
`DP4-70`	1.004	4.58

Based on the current observations, we can see that strict weak ordering $\prec_3$ is the worst performing in terms of cost ratio. Strict weak orderings $\prec_1$ and $\prec_2$ are comparable in terms of computation time and cost ratio. The best performing ordering, in terms of both computation time and cost ratio, is $\prec_4$.

Q5. What is the effect of applying phase synchronization in the dynamic programming heuristics?

Applying phase synchronization led to the following results.

method	cost	time (s)
`DP1-150-5`	1.006	16.76
`DP2-150-5`	1.011	14.84
`DP3-150-5`	1.023	13.73
`DP4-70-5`	1.003	4.92

When comparing the costs and times with those of the previous question, we can see that, in the current data, phase synchronization (in case $l = 2$) does not increase the computation time, while it does lead to slightly better cost ratios.

Q6. What is the effect of further gamete restriction in the dynamic programming heuristics?

The following results were obtained when further gamete restriction was applied.

method	cost	time (s)
`DP1-150-5g`	1.007	2.68
`DP2-150-5g`	1.011	1.82
`DP3-150-5g`	1.022	2.14
`DP4-70-5g`	1.004	0.61

In Subsection 5.1.4 we showed that further gamete restriction does not improve the asymptotic running time. Here we can see that the running times have become a lot less, while the cost ratios have remained the same. So the worst case scenario described in Subsection 5.1.4 does not occur (often) in practice.

Q7. Does a heterozygous ideotype lead to different answers on questions Q2-Q6?

To answer this question we consider dataset 6loc-het. Note that we did not run the randomized algorithm on this dataset, as it does not support a heterozygous ideotype.

Recall that in Subsection 5.1.2, we mentioned that in case of a heterozygous ideotype half of the genotypes retained are selected according to C_0^* and the other half according to C_1^*. In our implementation, however, we retain k genotypes according to C_0^* and another k genotypes according to C_1^*. So the number of genotypes retained in our implementation is actually $2k$. Because of this the computation times are higher than the ones of dataset 6loc-hom.

The answers on questions Q2-Q6 considering dataset 6loc-het are as follows.

(Q2) Again we can see in Figure 6.1a that the cost ratios of the dynamic programming heuristics are three times lower than the genetic algorithm. In Figure 6.1b, we can also see that the computation times of the dynamic programming heuristics are less than the one of the genetic algorithm. So the answer on this question is also yes for a heterozygous ideotype.

(Q3) Increasing the number of genotypes to be retained results in a lower cost ratio at the expense of a higher computation time.

(Q4) This time we have the following results.

method	cost	time (s)
`DP1-150`	1.000	54.28
`DP2-150`	1.004	50.10
`DP3-150`	1.004	34.60
`DP4-70`	1.000	14.71

Both $\prec_2$ and $\prec_3$ are the worst performing strict weak orderings in terms of the cost ratio. Again, strict weak orderings $\prec_1$ and $\prec_2$ are comparable in terms of computation time. Similarly to dataset 6loc-hom, $\prec_4$ is the best performing strict weak ordering in terms of computation time and cost ratio.

(Q5) Applying phase synchronization results in a slightly lower cost ratio, while at the same time the computation times do not change.

(Q6) Further gamete restriction results in similar cost ratios while the computation times decrease.

So the answers on questions Q2-Q6 remain the same in case of a heterozygous ideotype.

Q8. Does a different cost function lead to different answers on questions Q1 and Q2?

This question is answered using dataset 6loc-hom-pop. Using a cost function that only considers the population size, the cost ratio of the genetic algorithm is a lot better than with dataset 6loc-hom (1.220 as opposed to 3.530).

Similarly to dataset 6loc-hom, the randomized algorithm outperforms the genetic algorithm in terms of computation time and cost ratio. The same applies for the dynamic programming heuristics. The best performing method here is also `DP4-70-5g`, which achieves a cost of ratio of 1.012 and a mean computation time of 0.52 s.

Although the difference in cost ratios is a lot less dramatic, we still have that the randomized algorithm and the dynamic programming heuristics outperform the genetic algorithm. So the answer on this question is no.

Q9. Does a different number of loci lead to different answers on questions Q1 and Q2?

To answer this question we look at datasets 10loc-hom and 15loc-hom. In Figure 6.2, it can be seen that for both datasets the randomized algorithm outperforms the genetic algorithm in terms of computation time and cost ratio. The same applies for the dynamic programming heuristics. Again, the best performing method is `DP4-70-5g` with a cost ratio close to 1 and computations times of 5.85 s for ten loci and 39.04 s for fifteen loci. So the answer on this question is no.

Finally, we investigated how many times a method returned the best crossing schedule and also at how many times no result was produced by that method. Two plots in which the relative frequencies of these two quantities are depicted per dataset are given in Figure 6.3. We can see that the genetic algorithm does not return a result 40% of the time in case of a heterozygous ideotype. Also, we can see that the method that returns the best crossing schedule most often is `DP4-70-5g`.

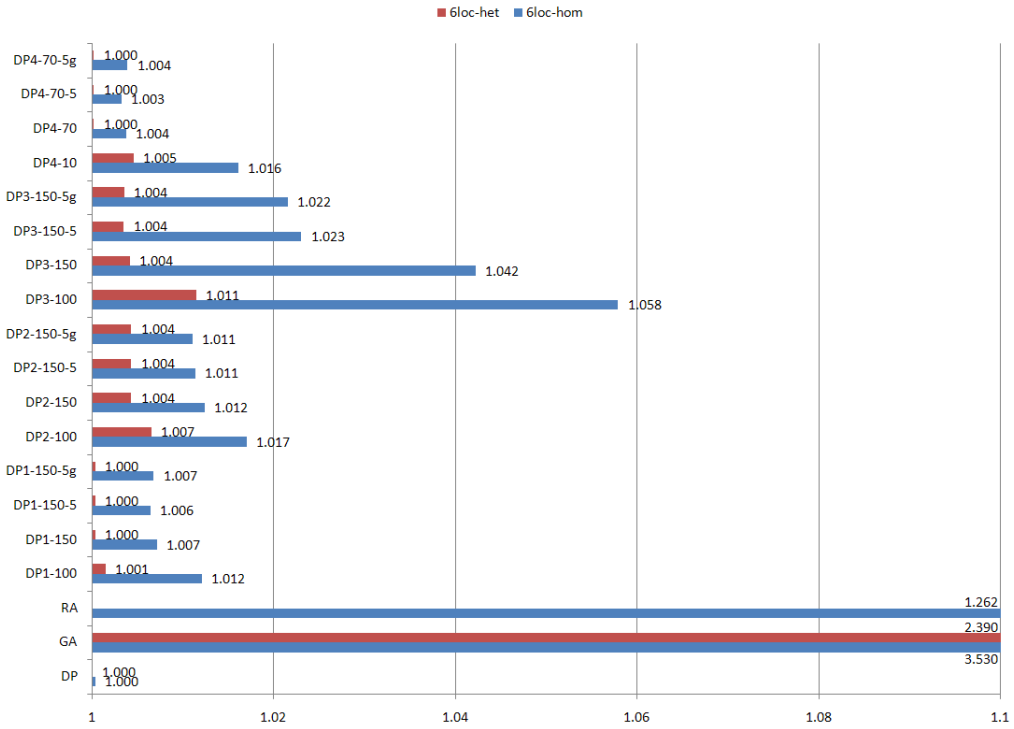
The conclusions that we can draw from all experiments are as follows.

- The randomized algorithm outperforms the genetic algorithm.
- The dynamic programming heuristics outperform the genetic algorithm.
- The genetic algorithm often fails to return a crossing schedule in case of a heterozygous ideotype.

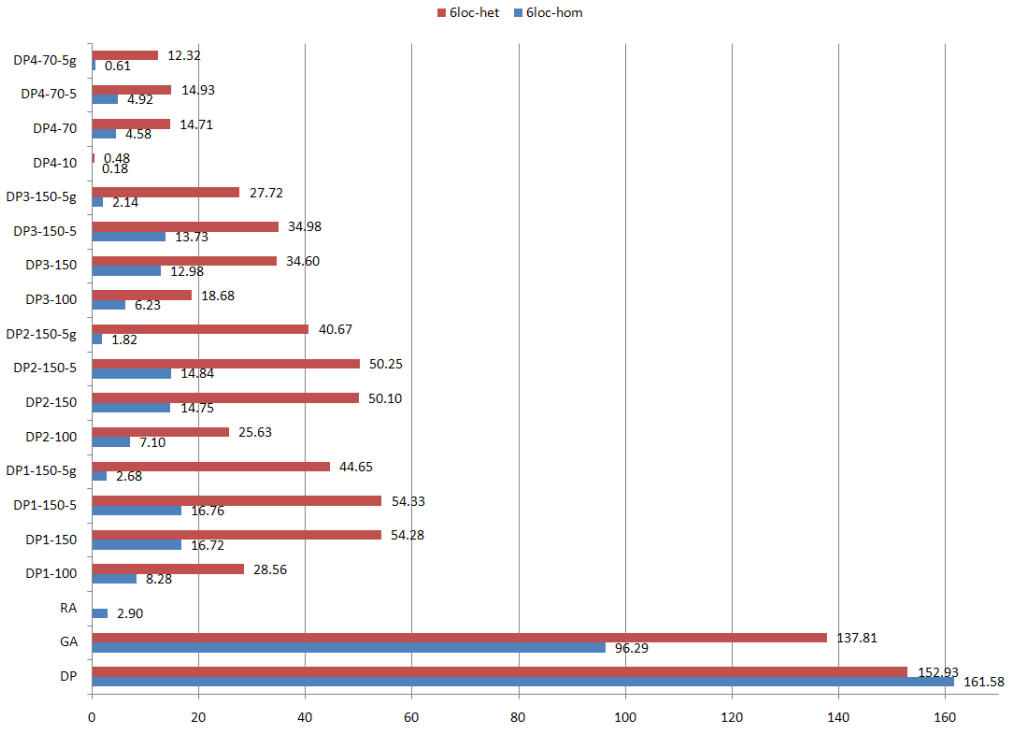
As for the dynamic programming heuristics, we can draw the following conclusions.

- Retaining a higher number of genotypes results in cheaper crossing schedules at the expense of larger computation times.
- The best performing strict weak ordering is $\prec_4$.
- Phase synchronization results in cheaper crossing schedules without having a (significant) effect on the computation times.
- Further gamete restriction results in a faster heuristic without (significantly) affecting the cost of a returned crossing schedule.

We have shown that the conclusions hold regardless of the number of loci considered and the cost function used. Also, we have shown that considering a heterozygous ideotype has no effect on the drawn conclusions.

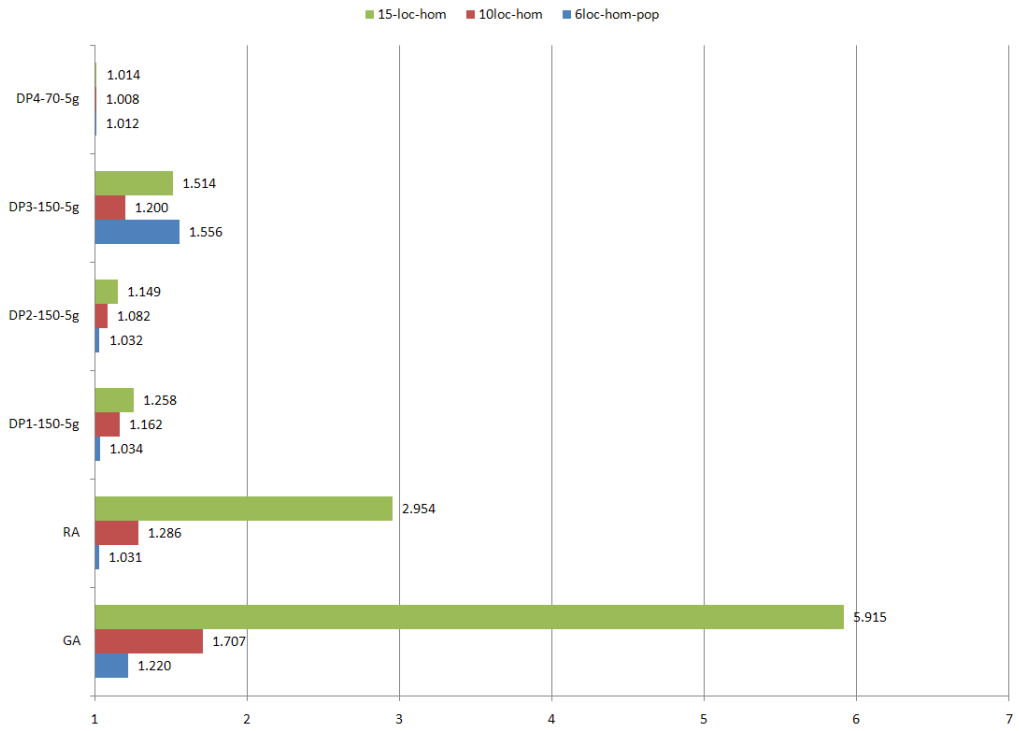


(a) Mean cost ratio (smaller is better). The cost ratio of a method expresses how more expensive the cost of a returned crossing schedule is compared to the lowest cost returned by other methods.

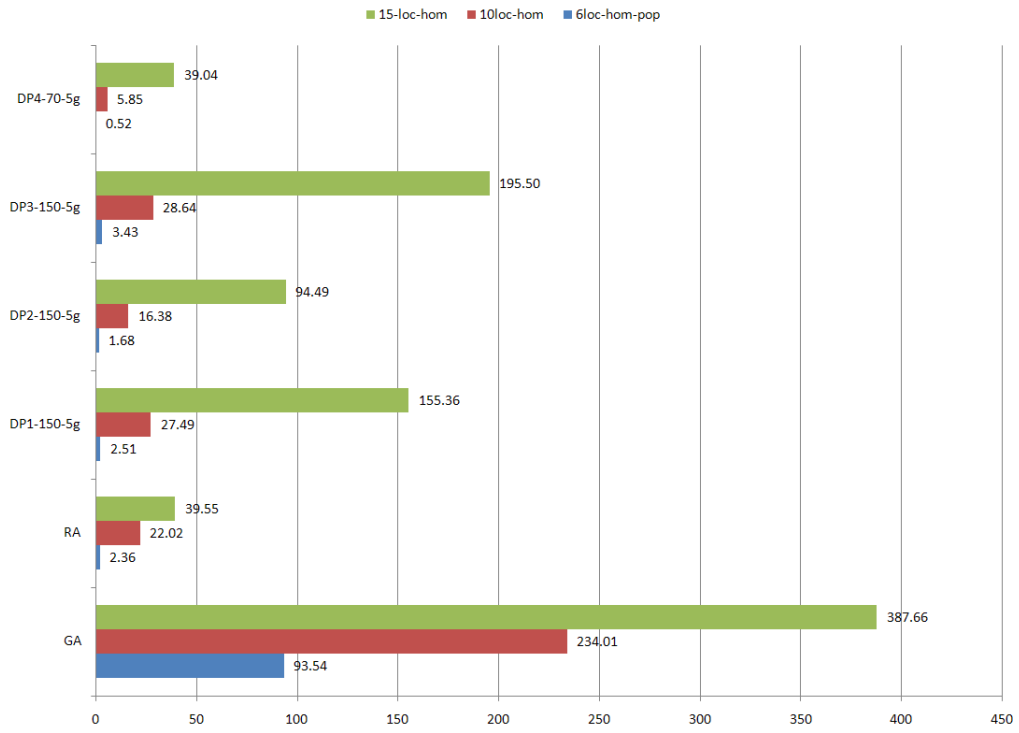


(b) Mean computation time in seconds (smaller is better).

Figure 6.1: Datasets 6loc-hom (blue) and 6loc-het (red).

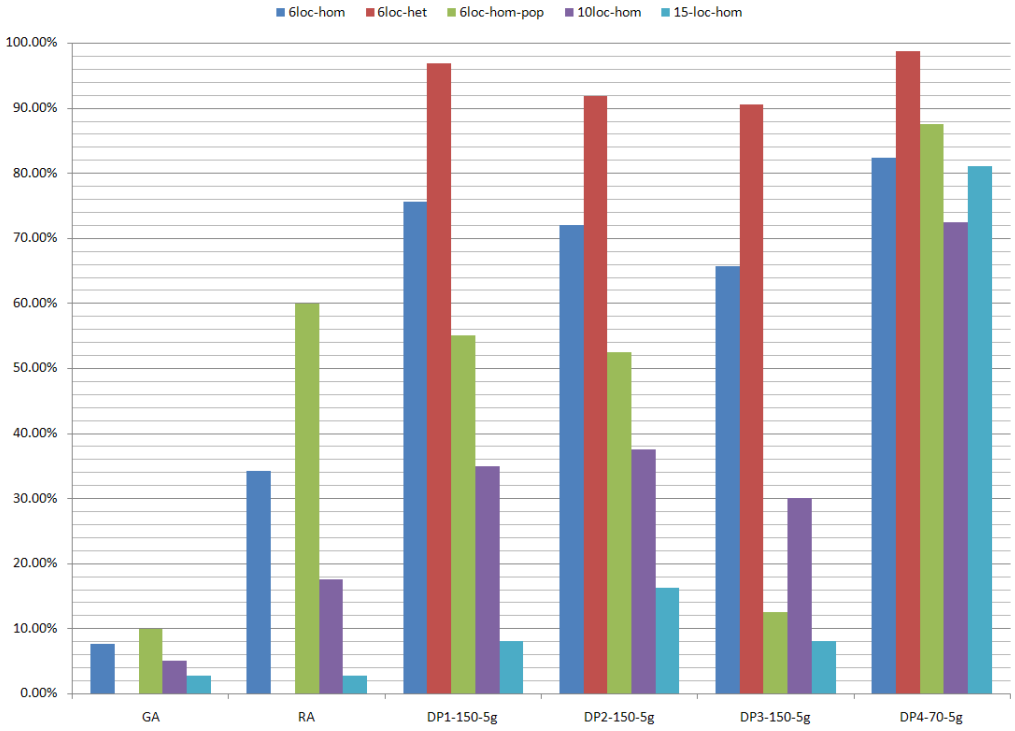


(a) Mean cost ratio (smaller is better). The cost ratio of a method expresses how more expensive the cost of a returned crossing schedule is compared to the lowest cost returned by other methods.

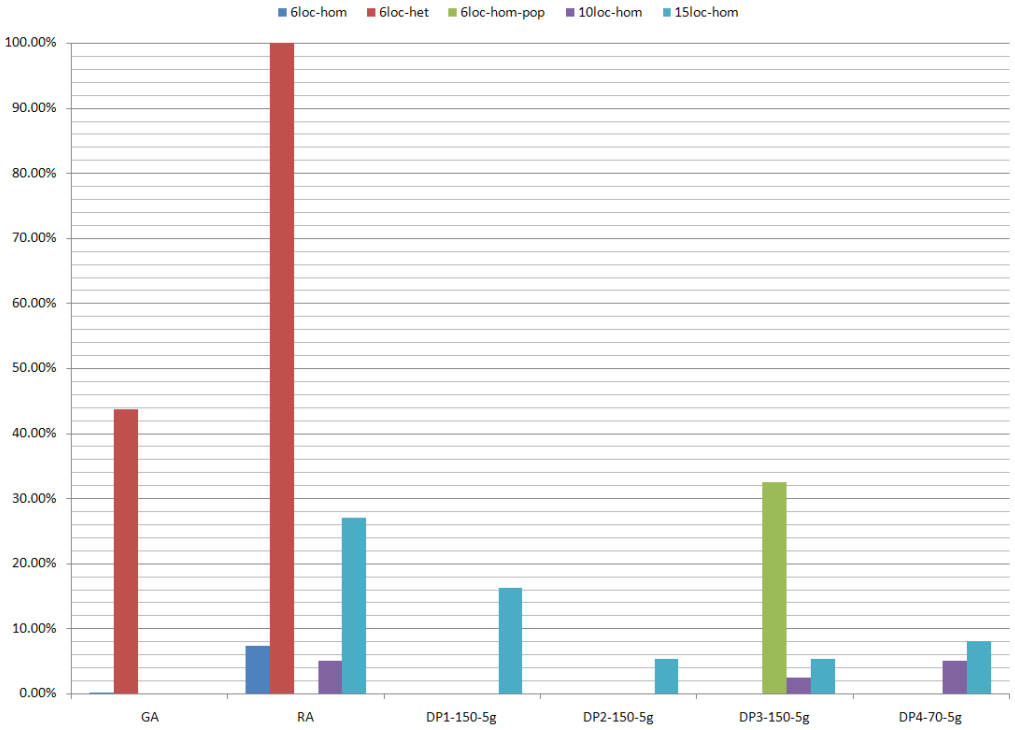


(b) Mean computation time in seconds (smaller is better).

Figure 6.2: Datasets 15loc-hom (green), 10loc-hom (red) and 6loc-hom-pop (blue).



(a) Relative frequency of problem instances on which the considered method performed the best.



(b) Relative frequency of problem instances on which the considered method returned no result.

Figure 6.3: Relative frequencies.

Chapter 7

Conclusion and Discussion

Over the years, numerous markers associated with desirable traits in crops have been identified. The ultimate objective of plant breeding is to obtain an individual having all the desired traits in a cost and time efficient way. The problem we have considered in this thesis is thus a relevant problem in plant breeding.

We have identified the number of generations, the number of crossings and the the total population size as the attributes used for assigning a cost to a crossing schedule. Using these attributes, we have defined the problem in a formal way in Chapter 2. To the best of our knowledge, this is the first time that the problem has been defined formally. In Chapter 3 we have shown that the problem is NP-hard. Also, we have shown that no constant factor approximation exists unless P = NP.

In Chapter 4 we have looked at existing methods. In the first section of this chapter, we have looked at a restricted version of the problem as presented by Servin et al.[16] We have analyzed the running time of their proposed algorithm and showed it to be $\mathcal{O}(n^{2n+3}2^{2n})$, where n is the number of parents. Subsequently, we have described a new method based on dynamic programming with a better running time of $\mathcal{O}(n4^n)$. In Section 4.2 we have described the genetic algorithm as it has been implemented within Keygene N.V. Subsequently, we have analyzed the performance of the algorithm and presented some recommendations. We have also discussed that for this problem it is very difficult to find an effective crossover operator. Therefore we have considered in Chapter 5 alternative methods.

The first method that we have presented in Chapter 5 is a heuristic based on dynamic programming. In Section 5.1, we have described the steps we took to arrive at a polynomial-time heuristic. Subsequently, we have described additional methods aimed at improving the running time (in practice) and obtaining lower-cost crossing schedules. One way of improving the running time in practice is based on the proof that homozygous genotypes can be obtained optimally via selfing. In Section 5.2 we have described a randomized algorithm that only works for heterozygous ideotypes.

In Chapter 6 we have experimentally evaluated the genetic algorithm, the randomized algorithm and the dynamic programming heuristic. For the dynamic programming heuristic we have looked at the effect of the different parameters that we have introduced in Section 5.1. The main conclusion is that both the randomized algorithm and the dynamic programming heuristic outperform the genetic algorithm in terms of computation time and cost of a returned schedule.

7.1 Open problems and future work

The NP-hardness proof that we have given in Chapter 3 only considers the number of crossings in the cost function. From a theoretical point of view, it is also interesting to know whether the problem with a cost function that only considers the population size is NP-hard as well. Unfortunately, we do not know if this is the case. If this problem turns out to be NP-hard, it is interesting to know whether this time a constant factor approximation exists. On the other hand, if a polynomial-time algorithm for the problem would exist, finding a crossing schedule that is guaranteed to be optimal would become tractable.

Using parallelization, we can make our methods run even faster. For instance, if we require ten hours to compute a schedule for a certain problem instance, it would be possible to compute the same schedule in one hour using a cluster consisting of ten nodes. The dynamic programming heuristic that we have presented in Section 5.1 can easily be parallelized. The first step towards obtaining a parallel algorithm is deciding on the memory model to use [1]. We suggest to use a distributed memory model such that every processor contains a complete copy of the dynamic programming table. The next step in obtaining a parallel algorithm is to decide on how the work that is done per iteration should be distributed among the processors. In our case most of the work done in an iteration consists of performing pairwise crossings among the genotypes in two disjoint lists. When we view the pairwise crossings that are performed as a matrix, we can distribute the work by assigning in a cyclic fashion the rows to the processors. Of course, when actually developing and implementing a parallel algorithm much more details have to be taken care of.

Bibliography

[1] R. H. Bisseling. *Parallel Scientific Computation*. Oxford University Press, 2004.

[2] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[3] J. Brown and P. Caligari. *Introduction to Plant Breeding*. Wiley-Blackwell, 1st edition, 2008.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2001.

[5] J. C. M. Dekkers and F. Hospital. The use of molecular genetics in the improvement of agricultural populations. *Nature Reviews Genetics*, 3:22–32, 2002.

[6] A. J. F. Griffiths, J. H. Miller, D. T. Suzuki, R. C. Lewontin, and W. M. Gelbart. *An Introduction to Genetic Analysis*. W. H. Freeman and Company, eight edition, 2005.

[7] J. B. S. Haldane. The combination of linkage values and the calculation of distances between the loci of linked factors. *Journal of Genetics*, 8:299–309, 1919.

[8] T. Ishii and K. Yonezawa. Optimization of the marker-based procedures for pyramiding genes from multiple donor lines: I. Schedule of crossing between the donor lines. *Crop Science*, 47:537–546, 2007.

[9] T. Ishii and K. Yonezawa. Optimization of the marker-based procedures for pyramiding genes from multiple donor lines: II. Strategies for selecting the objective homozygous plant. *Crop Science*, 47:1878–1886, 2007.

[10] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[11] D. D. Kosambi. The estimation of map distances from recombination values. *Annals of Eugenics*, 12(3):172–175, 1944.

[12] D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, 4th edition, 2006.

[13] S. P. Moose and R. H. Mumm. Molecular plant breeding as the foundation for 21st century crop improvement. *Plant Physiology*, 147:969–977, 2008.

[14] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 475–484, 1997.

[15] J. F. Rohlf. Numbering binary trees with labeled terminal vertices. *Bulletin of Mathematical Biology*, 45(1):33–40, 1983.

[16] B. Servin, O. C. Martin, M. Mézard, and F. Hospital. Toward a theory of marker-assisted gene pyramiding. *Genetics*, 168(1):513–523, 2004.

[17] C. Shifriss, M. Pilowsky, and J. M. Zacks. Resistance to Leveillula Taurica mildew (=Oidiopsis taurica) in Capsicum annuum. *Phytoparasitica*, 20(4):279–283, 1992.