Aalborg University

Department of Computer Science, Fredrik Bajers Vej 7E, DK-9220 Aalborg Øst

TITLE:
A New Approach to Multi Point Linkage Analysis - Decision Diagrams Applied in Linkage Analysis

TOPIC:
Distributed Systems and Semantics, Linkage Analysis

PROJECT GROUP:
E1-110b

MEMBERS OF PROJECT GROUP:

Anders Lyhne Christensen

Jacob Illum Rasmussen

SUPERVISOR:
Anna Ingólfsdóttir

SEMESTER:
DAT6

PROJECT PERIOD:
February 1st to June 2002

NUMBER OF COPIES:
5 (1 online)

NUMBER OF PAGES:
142

Synopsis:

This report documents research aimed at developing a new heuristic-based algorithm for multi point probability computation in linkage analysis. A short introduction to the underlying biology, and a summary of a previously developed formal framework, are provided.

An introduction to multi point linkage analysis and the methods behind one of the fastest, current algorithms based on the fast Fourier transform, are included.

A new algorithm is proposed, the recombination transform, and extended in order to make it applicable to probability distributions over founder reduced inheritance vectors.

The results of an evaluation of the applicability of seven decision diagram-based data structures for compact representations of probability distributions in linkage analysis, are presented. It is concluded that data structures based on the Shannon decomposition can provide such compact representations.

Finally, an algorithm which performs the recombination transform on MTBDDs is proposed and its correctness proven. Experimental results of a prototype implementation, using this algorithm and MTBDDs for representation, are presented. Multi point probabilities are successfully computed for previously prohibitively large pedigrees.

Contents

List of Symbols

P	Pedigree $P = \langle F, N, father, mother \rangle$, see Definition 1 on page 16.
$father$	Father function $father : N \to V$, see Definition 1 on page 16.
$mother$	Mother function $mother : N \to V$, see Definition 1 on page 16.
V	Set of individuals in a pedigree (P), $V = N \cup F$, see Definition 1 on page 16.
F	Set of founders in a pedigree (P), $F \subset V$, see Definition 1 on page 16.
N	Set of non-founders in a pedigree (P), $N \subset V$, see Definition 1 on page 16.
$ancestor$	Ancestor function $ancestor : V \to 2^V$, see Definition 1 on page 16.
A	Set of alleles for a pedigree, see Definition 2 on page 17.
M	A marker $M = \langle A, \pi \rangle$, see Definition 3 on page 18.
A	The set of allelic states for the marker M, see Definition 3 on page 18.
π	A function $\pi : A \to [0, 1]$, maps the allelic states of A to its allele frequency, see Definition 3 on page 18.
$\mathcal{G}$	Genotype information $\mathcal{G} = \langle \mathcal{L}, astates \rangle$, see Definition 4 on page 18.
$\mathcal{L}$	Set of genotyped individuals for a pedigree, see Definition 4 on page 18.
$astates$	Allelic state function $astates : \mathcal{L} \to \{\{a_1, a_2\} \mid a_1, a_2 \in A\}$, see Definition 4 on page 18.
v	Inheritance vector function: $v : N \times \{p, m\} \to \{p, m\}$, see Definition 5 on page 19.
$\mathbf{v}$	The set of all inheritance vectors for a pedigree, see Section 3.2.2 on page 21.
p	Parameter for the inheritance vector function, denotes the *paternal* allele, see Definition 5 on page 19.
m	Parameter for the inheritance vector function, denotes the *maternal* allele, see Definition 5 on page 19.
Z	A founder allele assignment, $Z : F \times \{p, m\} \to A.$ for some marker M, see Definition 6 on page 19.
F^v	Allele assignment function $\mathsf{F} : V \times \{p, m\} \to F \times \{p, m\}$, which given an inheritance vector gives the founder alleles received by an individual on the maternal or paternal side, see Definition 7 on page 20.
$P(v \mid \mathcal{G})$	The probability of the inheritance vector v given some genotype information $\mathcal{G}$, see Section 3.2.2 on page 21.
$P(\mathcal{G} \mid v)$	The probability of some genotype information $\mathcal{G}$ given an inheritance vector v, see Section 3.2.2 on page 21.
$P(Z)$	The probability of the founder allele assignment, see Section 3.2.2 on page 21.
$\mathcal{F}^v_{\mathcal{G}}$	The set of all compatible founder allele assignments given the inheritance vector v, and the genotype information $\mathcal{G}$, see Section 3.2.2 on page 21.
$a_1, a_2, \ldots$	Allelic states.
ϖ	Denotes either p for paternal or m for maternal ($\varpi \in \{p, m\}$).

P	Probability vector over complete inheritance vectors, see Example 12 on page 24.
θ_i	The recombination fraction between markers M_i and M_{i+1}, see Definition 9 on page 24.
$Ham(v, w)$	The Hamming distance between two inheritance vectors v and w, see Definition 10 on page 25.
T	Transition matrix between complete inheritance vector at different markers, see Equation 3-4 on page 26.
C	Contribution vector. The contribution from all inheritance vectors at one marker to all inheritance vectors at another marker ($C = T \cdot P$), see Equation 3-5 on page 26.
C_i^L	Contribution to the inheritance vectors at one marker from all markers located to the left, see Equation 3-7 on page 28
C_i^R	Contribution to the inheritance vectors at one marker from all markers located to the right, see Equation 3-8 on page 28
P^{mp}	Multi point probability vector, see Equation 3-9 on page 28.
P^{sp}	Single point probability vector, see Equation 3-9 on page 28.
$\mathbb{Z}_2$	The set of integers modulo 2, that is $\mathbb{Z}_2 = \{0, 1\}$, see Section 3.3.2 on page 30.
$\mathbb{Z}_2^n$	The direct product of n $\mathbb{Z}_2$, see Section 3.3.2 on page 30.
$\oplus, \ominus, \odot$	Bitwise addition, subtraction, and scalar product over $\mathbb{Z}_2^n$, see Section 3.3.2 on page 30
$f, \hat{f}$	$\hat{f}$ is the Fourier transform of the function f, see Theorem 1 on page 31.
$\widehat{P}, \widehat{T}, \widehat{C}$	Fourier transforms of P, T, and C.
W_n	Walsh-Hadamard transform of order n, see Definition 11 on page 33.
$\otimes$	Kronecker product between matrices, see Equation 3-19 on page 34.
$P_{\overline{x}_i}, P_{x_i}$	Negative and positive cofactors of a vector. Sometimes referred to as P_0 and P_1 if the variable is understood or unimportant, see page 34.
$\mathcal{R}_n^\theta$	Recombination transform of order n over recombination fraction θ, see Definition 12 on page 39.
o	Bijection of inheritance vector layout, $o : N \times \{p, m\} \rightarrow \{1, \ldots, 2 \cdot \lvert N \rvert\}$, see Definition 13 on page 42.
s_h	Founder bit mask for some founder $h \in F$, see Definition 14 on page 42.
S	Subgroup of $\mathbb{Z}_2^n$ generated by the set of founder bit masks, see Theorem 6 on page 43.
$\llbracket v \rrbracket_S$	Equivalence class of inheritance vector v under S, see Theorem 6 on page 43.
$\widetilde{P}$	Probability vector over founder reduced inheritance vectors, see Equation 4-6 on page 46.
v_x	The complete inheritance vector, $v_x \in \mathbb{Z}_2^n$ which is the representative element of the equivalence class represented by the founder reduced inheritance vector $x \in \mathbb{Z}_2^{n-f}$, see page 47.
$\widetilde{T}$	Transition matrix between founder reduced inheritance vectors, see page 49.

$\langle q, I\rangle$	Mutation tuple, q is the number of founders for which bits have been switched and I is the set of indices which have been switched, see Definition 15 on page 50.
$\mathcal{I}$	Set of mutation tuples, see Definition 15 on page 50.
$\overline{\mathcal{R}}_n^\theta$	Reverse recombination transform of order n over recombination fraction θ, see Definition 16 on page 52.
$\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}$	Hybrid recombination transform of order n over recombination fraction θ and mutation tuple $\langle q, I\rangle$, see Definition 17 on page 53.
$\widetilde{C}$	Contribution for founder reduced inheritance vectors, see Theorem 7 on page 55.
Γ	Set of indices for bits in an inheritance vector which represent meiosis unrelated to founders, see page 59.
α	Set of indices for all the first bits in an inheritance vector which represent a meiosis of a founder, see page 59.
β	Set of indices for bits in an inheritance vector which represent meiosis unrelated of founders, but not the first bit, see page 59.
$\Re_n^\theta$	Founder reduced recombination transform of order n over recombination fraction θ, see Equation 4-18 on page 60.
$\Re_n^{\theta,nor}$	Founder reduced recombination transform of order n over recombination fraction θ for decomposition with respect to the normal recombination transform, see Equation 4-18 on page 60.
$\Re_n^{\theta,ref}$	Founder reduced recombination transform of order n over recombination fraction θ for decomposition with respect to the reverse recombination transform, see Equation 4-18 on page 60.
MTBDD(g)	MTBDD representing function g, see Definition 18 on page 64.
VAR	Set of variables in a MTBDD, see Definition 18 on page 64.
V	Set of vertices in a MTBDD. Not to be confused with the of all individuals in a pedigree. The meaning should be clear from the context. see Definition 18 on page 64.
T	Set of terminals in a MTBDD. Not to be confused with the of all individuals in a pedigree. The meaning should be clear from the context. see Definition 18 on page 64.
$level$	Function which returns the level of a node in a MTBDD $level : V \cup T \rightarrow \mathbb{N} \cup \{\infty\}$, see Definition 18 on page 64.
$label$	Function which returns the variable described by an node in a MTBDD, if the node is a terminal the value of the node is returned, $label : V \cup T \rightarrow$VAR$\cup\mathbb{R}$, see Definition 18 on page 64.
$child_0$	The negative child of a node in a MTBDD, $child_0 : V \rightarrow V \cup T$, see Definition 18 on page 64.
$child_1$	The positive child of a node in a MTBDD, $child_1 : V \rightarrow V \cup T$, see Definition 18 on page 64.
$g_{\overline{x}_i}, g_{x_i}$	Negative and positive cofactors of some function, g, see Example 39 on page 117.
$f(\langle n\rangle)$	Function over node in a MTBDD, see Example 34 on page 66.
BFWG	Breadth-first with grouping variable ordering, see Section 5.5 on page 72.

BFNG	Breadth-first no grouping variable ordering, see Section 5.5 on page 72.
DF	Depth-first variable ordering, see Section 5.5 on page 72.
$children$	Returns the set children of a pedigree member, $children : F \cup N \to 2^N$, see Definition 19 on page 76.
$sibs$	Returns the set of siblings of a non-founder in the pedigree, $sibs : N \to 2^N$, see Definition 20
DH	Descendant height. A measure of how far an individual is the its most distant descendant, $DH : V \to \mathbb{N}$, see Definition 21 on page 76.

Introduction

CHAPTER 1

Linkage analysis is a method used to locate genes in the human genome, which are responsible for various traits. This method has major applicability in the pharmaceutical industry, since it can assist in the development of diagnostic tools and drugs for diseases influenced by genetic factors.

The core of linkage analysis is to determine whether one or more disease genes are located close to a known gene, a marker. If a gene, influencing or causing some disease, is located close to a marker on the genome, this gene and the marker are said to be *linked*, hence the name *linkage analysis*. Linkage can be detected by analyzing whether or not a disease and a marker have matching inheritance patterns in pedigrees. Depending on the data, every possible inheritance patterns might have a probability of being the true inheritance pattern. This induces some limits to the size of the pedigrees which can be analyzed, since the space and time complexity of most algorithms currently used are exponential in the pedigree size.

The size of a pedigree is measured in bits, not just because the majority of the people involved in the field have some romantic relationship with modern technology, but simply because it is a convenient way to express inheritance patterns. In recent years various attempts have been made to push the limit, however, the limit has only been moved a few bits at the time: In year 2000 Gudbjartsson, the author of the linkage analysis software Allegro, [Gud00] identified symmetries in pedigrees under some circumstances which lead to a reduction called the *founder couple reduction*. The limit was pushed to pedigrees of the size 24 to 25 bits, which corresponds to pedigrees of about 27 individuals depending on the pedigree structure. In 2001 Kruglyak et al. discovered that certain inheritance patterns can be ignored dependent on the data observed, a method referred to as *fixing bits*, [MDK01]. Limit: 30 bit pedigrees (about 35 individuals). In early 2002 Abecasis et al. showed that the use of *sparse gene flow trees* can push the limit to 32 bits (about 38 individuals), [ACCC02].

The purpose of the work presented in this report is to develop an alternative method for representation of probability distributions over inheritance patterns, and a new algorithm for the part of linkage analysis referred to as *multi point probability computation*. In this part of the analysis the purpose is to compute the probabilities of inheritance patterns based on data from multiple markers (or points) across the human genome. The multi point probability computation is one of, if not *the*, limiting task of linkage analysis, [MDK01, Gud00]. Our ambition is not to improve the current methods, our aim is to approach the problem from a different angle.

Linkage analysis is a field populated with researchers of from different scientific disciplines, such as biology, statistics, and computer science. Since we have a background in computer science we have focused on the representation of data and on improving the algorithms, while remaining true to the biological and statistical models. For the representation of large data sets, that is probability distributions over inheritance patterns in linkage analysis, we investigate the applicability of *decision diagram*-based heuristic methods. Decision diagrams (DDs) are data structures for graph-based representations of discrete functions. For some domains, such as formal model checking, DDs have provided compact representations of large state spaces, [KKNP01, KNPS99]. We investigate whether DDs have anything to offer in linkage analysis, and if efficient algorithms for multi point probability computation can be developed for such data structures.

1.1 Outline

We have organized the report as follows:

In Chapter 2 we go through some of the biology behind linkage analysis. Since it is essential to understand the basic concepts and motivations behind the method, we urge readers unfamiliar with molecular biology to read this chapter. Since we use many of the terms presented in Chapter 2 throughout the rest of the report, we have supplied a glossary for convenience. The glossary is found in the back of the report.

In Chapter 3 we first summarize the formal model developed in our previous work, [CHI+02]. We then present the motivation behind the part of linkage analysis called *single point probability computation*. Single point probability computation is the part of linkage analysis where probabilities of different inheritance patterns are computed based on data for *single* markers. We have omitted any algorithms for single point analysis, and we refer interested readers to our previous work [CHI+02] for the presentation of two such algorithms. We proceed with the idea behind *multi point probability computation*, and go into some theoretical depth with respect to the algorithms currently used in two modern linkage analysis packages, namely GeneHunter and Allegro, [Gud00, MDK01, KL98].

In Chapter 4 we propose for a new, faster algorithm for multi point probability computation, called the *recombination transform*.

In Chapter 5 we investigate the applicability of a number of decision diagram-based data structures, to determine if such structures can provide more compact representations of probability distributions in linkage analysis.

In Chapter 6 we state and prove two algorithms for performing the recombination transform, the algorithm proposed in Chapter 4, on Multi Terminal Binary Decision Diagrams. We provide experimental results of our method, discuss how the running time is related to the input data, and how it performs compared to Allegro, the linkage analysis software currently used at deCODE Genetics.

Finally, in Chapter 7 we provide a summary of the report and our contribution to linkage analysis, followed by a list of topics which we consider interesting for further research.

1.2 How to Read This Report

This report is primarily written for readers with a background in computer science. However, it is intended that most people with an interest in linkage analysis should be able to understand and benefit from the results of this report. The report can be read in a number of ways depending on the readers intension.

Chapters 2 and 3 can serve as an introduction to the underlying biology, as well as single and multi point analysis. Readers familiar with multi point linkage analysis can skip Chapters 2 and 3, and move directly to Chapter 4 where we derive the recombination transform. Chapter 5 should be read by readers who are interested in the performance of different decision diagram-based data structures in the domain of linkage analysis. For less keen readers it suffices to read the summary of the chapter.

Most of Chapter 6, in which we present an algorithm for the recombination transform on MTBDDs and experimental results, requires that the reader is familiar with the theory behind the recombination transform and decision diagram-based data structures. However, readers who are simply interested in results should take a peak at the plots of the running times of a prototype implementation of the recombination transform on MTBDDs.

This report is 113 pages long (excluding appendices), which must be considered above average for a master thesis from the computer science department of Aalborg University. However, since we expect the primary reader to have a background in computer science - and not in statistics or biology - some space has been devoted to examples and background material to make the report as self-contained as possible.

1.3 Acknowledgments

First, we would like to acknowledge Jens Hansen, John Knudsen, and Jacob Johnsen for our previous collaboration in developing the formal framework for linkage analysis presented in [CHI+02].

We would like to acknowledge Daniel Fannar Gudbjartsson, the creator of Allegro, for his help and guidance as well as other people in the Statistics Department, and Sverrir Þorvaldsson and Gisli Màsson, at deCODE Genetics for their patience and willingness to answer questions and give explanations.

We would like to thank the following people for inspiration: Professor of Computer Science Kim G. Larsen and Associate Professor of Computer Science Luca Aceto, both from Aalborg University. Professor of Biotechnology Aalborg University Karen G. Welinder, Assistant Professor Dennis Nilsson from the Mathematics Department at Aalborg University, Research Assistant Professor at BRICS Aarhus Christian N. Storm Pedersen, Professor of Computer Science at the University of Birmingham Martha Kwiatkowska and Ph.D. student at the University of Birmingham David Parker.

Biology

Chapter 2

This chapter introduces the reader to the terms and concepts of human genetics which are essential for understanding linkage analysis. Readers who are familiar with genetic concepts like alleles, mendelian inheritance, linkage analysis, and so on may want to skip this chapter. We remind the reader unfamiliar with any of these concepts, that there is a glossary on page 141, where it is possible to find a short explanation and a reference to many of the terms and concepts present in this chapter.

The structure is as follows. We begin with a rather high level description of genetics, by introducing the two major concepts of chromosomes and DNA. This is followed by the introduction of alleles and allelic states, which describe the different forms a gene can assume. Two important concepts in inheritance, meiosis and recombination, are introduced. Furthermore, we introduce the concept of linkage analysis and finally summarize the chapter.

2.1 Chromosome

Inside each human cell there are 23 pairs of chromosomes, 22 pairs of *autosomes* common for both males and females, and one pair of sex-chromosome which differs between the two sexes. A chromosome consists of two identical sister *chromatides* - each a double helix DNA, *strand*, [SR99, page 30], thus a total of 92 DNA strands, constituting 46 chromosomes. During fertilization a fetus receives one chromosome from each parent (actually it is one chromatide, which later is transformed into a chromosome, but in this context it can be thought of as a chromosome).

The complete set of genetic material is called *The Human Genome*.

2.2 DNA

The molecule that encodes genetic information of humans is deoxyribonucleic acid (DNA). DNA is a double helix molecule that consists of a sugar-phosphate backbone and four nitrogenous bases - adenine, thymine, cytosine, and guanine (A, T, C, and G), [SR99, Chapter 1]. The nitrogenous bases are also called *base pairs* (bps), as they always pair: A with T, and C with G. The base pairs are situated between the two helixes, thereby binding the two backbones together.

A combination of three bps are termed a *codon*[1]. Each codon specifies an amino acid or a stop code (see Figure 2-1 on the following page). After a stop code, the bps can possibly specify

[1] Also termed a *triplet.*, [KC97, page 325].

a non-coding region, which for example could be hundreds of C-T combinations. Thus, 3 bps code a codon, codons code the sequence of amino acids, which again determines the genetic code for a single protein molecule (a gene), [Lan97, page 246]. *Proteins* function as enzymes, hormones, etc. and therefore influence many physiological and psychological *traits*, e.g. eye-color and schizophrenia. Thus, DNA is the recipe for proteins where the genetic code consists of nitrogenous bases interpreted in sequences of three. When the genetic code is translated to a protein, each codon specifies an amino acid. Amino acids are the building blocks of proteins. When a stop codon is reached the translation process is ended and the protein is finished and ready to perform its function, such as to signal other cells to start or stop the production of enzymes, in case the protein just build is a hormone.

Some traits are more dependent on outside stimuli than others, e.g. some traits might not show at all, due to the environmental factors, while other traits only show in the presence of one or more outside stimuli. For example, it is believed that a gene exists, that predisposes individuals for obesity[2]. However, the obesity trait is only shown in the case where plentiful nutrients have been available for an individual with the disease gene.

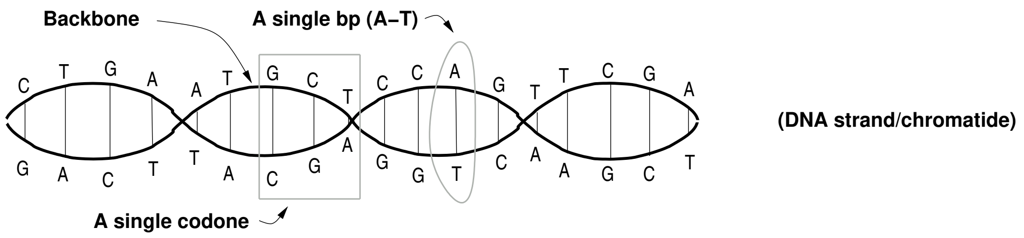

Figure 2-1: A subsection of a chromatide, which is also known as a (subsection of a) strand. The molecular structure is a DNA molecule. Notice the pairing A with T and C with G composing the 18 bps, that are coding 6 codons.

The sequence of base pairs encoding a single protein is called a *gene*, and the section or position on the chromosome harboring a gene is denoted as the *locus* (plural: loci) of the gene.

A well characterized locus can serve as a genetic *marker*, by well characterized we mean that the locus of the gene is known and that the different allelic states of the gene are known (allelic states will be explained shortly). Markers can be used as reference points on a chromosome and they can be analysed in order to determine the genotype of an individual at a given marker, [Gud00, page 9].

2.3 Alleles

An individual has two alleles at each locus of a gene, since one chromosome is received from each parent. An *allelic state* refers to a specific encoding (in codons) of a specific allele. The allelic states on the two chromosomes are not necessarily different, but they can be, hence the

[2]We learned that a group of statisticians and biologists are currently working on isolating the gene for this disease, during our stay at deCODE Genetics. Some results have already been released as news, [dNC01].

sequences of codons coding the alleles, and thereby genes, are different. An individual is said to be *homozygous* if the *maternal* and the *paternal* allelic states (the allele received from the mother and the allele received from the father) at a locus are identical. If the allelic states are different, the individual is said to be *heterozygous* at that locus.

Two alleles at a locus make up the *genotype* of that locus for a specific individual. If a person is heterozygous it is possible that only one of the alleles are *expressed*. In this case, the allele expressed is called *dominant*, while the allele not expressed is called *recessive*. If both alleles are expressed, they are said to be *codominant*. The expression of a genotype is called the *phenotype*.

Example 1 *As an example consider the human blood type. The gene for the human blood type is located on chromosome 9 at band q34, [Lan97, page 2] (band refers to a position on the chromosome). Three allelic states exists for blood type:* $\mathbf{A}$, $\mathbf{B}$, *and* $\mathbf{0}$. *Each person has two alleles, however there exists only four blood types*[3] *(phenotypes):* $\mathbf{A}$, $\mathbf{B}$, $\mathbf{A}/\mathbf{B}$, *and* $\mathbf{0}$, *even though there are are six possible combinations of allelic states:* $\mathbf{A}/\mathbf{A}$, $\mathbf{B}/\mathbf{B}$, $\mathbf{0}/\mathbf{0}$, $\mathbf{A}/\mathbf{B}$, $\mathbf{A}/\mathbf{0}$, *and* $\mathbf{B}/\mathbf{0}$ *(note that the ordering is irrelevant). If an individual has the allelic states* $\mathbf{A}/\mathbf{A}$, $\mathbf{B}/\mathbf{B}$, *or* $\mathbf{0}/\mathbf{0}$ *the individual is homozygous. In the event that an individual has the allelic states* $\mathbf{A}/\mathbf{0}$ *the individual is heterozygous, and for these alleles* $\mathbf{A}$ *is dominant and* $\mathbf{0}$ *recessive, since only the* $\mathbf{A}$ *allele is expressed, giving the person the* $\mathbf{A}$ *blood type, just as if the person had the genotype* $\mathbf{A}/\mathbf{A}$. $\mathbf{B}$ *is also dominant with respect to* $\mathbf{0}$, *since a person will have blood type* $\mathbf{B}$ *if the person has either the genotype* $\mathbf{B}/\mathbf{B}$ *or* $\mathbf{B}/\mathbf{0}$. *However,* $\mathbf{A}$ *and* $\mathbf{B}$ *are codominant, since an individual with the genotype* $\mathbf{A}/\mathbf{B}$ *will have neither blood type just* $\mathbf{A}$ *or* $\mathbf{B}$, *but* $\mathbf{A}/\mathbf{B}$. *The final blood type* $\mathbf{0}$ *is only expressed if a person has the genotype* $\mathbf{0}/\mathbf{0}$.

2.4 Meiosis

In this section we present the meiosis in a simplified version, however, sufficient for understanding the later chapters. Due to the complexity of this biological process we consider it out of scope to explain it in detail[4].

Meiosis is an essential process of the human reproduction. Abstractly, it is a cell division process that produces *gametes*, female egg cells or male sperm cells, which combine in reproduction and become the seed for a new individual. Meiosis is a process where a cell with one chromosome pair (a *diploid* cell) is divided into four cells with one chromatide in each (four *haploid* cells). This is illustrated in the left half (before the second $\Rightarrow$) of Figure 2-2 on the next page.

The probability that a given gamete is passed on to a child is assumed to be random. Hence, probability $\frac{1}{4}$ for each of the four combinations. A single gamete is randomly "selected" for reproduction (see the right half of Figure 2-2 on the following page). Since a given allele is present in precisely two of the four gametes, the probability that an allele being transmitted to an offspring is $\frac{1}{2}$ (from $2 \cdot \frac{1}{4}$). This hypothesis, that the source of inheritance for a single allele

[3] For simplicity we have omitted the fact, that a person can either be resus negative or resus positive.
[4] See [KC97, Chapter 2] for a detailed description of meiosis.

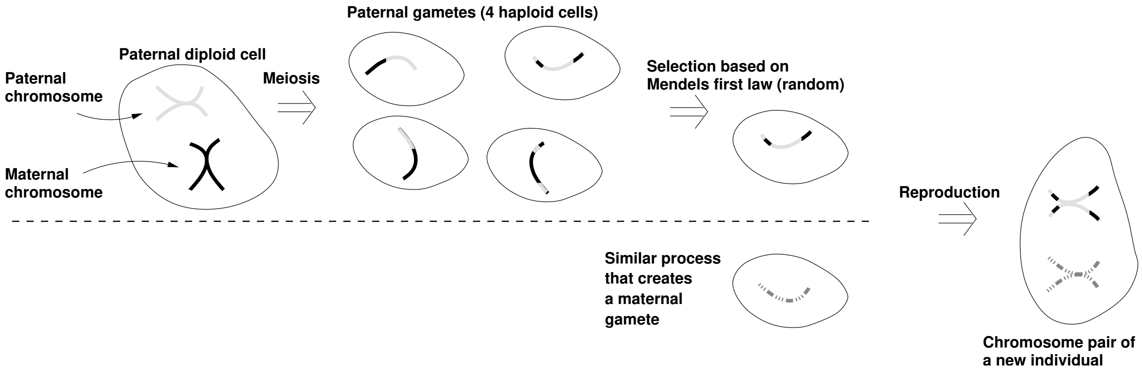

Figure 2-2: Meiosis illustrated for a single chromosome pair of a male and (partially) a female. The results of their meiosis is later combined in reproduction to form the chromosome pair of a new individual. Note that this is simplified, in reality there are 22 more chromosomes in each cell, to which a similar process takes place in parallel.

is equally likely to be either of the two grandparents, was stated by Gregor Johann Mendel in the 18'th century and is known as *Mendel's first law*, [KC97, Chapter 3].

The pair of alleles received from both the parents at a given locus constitute a child's genotype at that given locus. As an example see the pedigree in Figure 2-3.

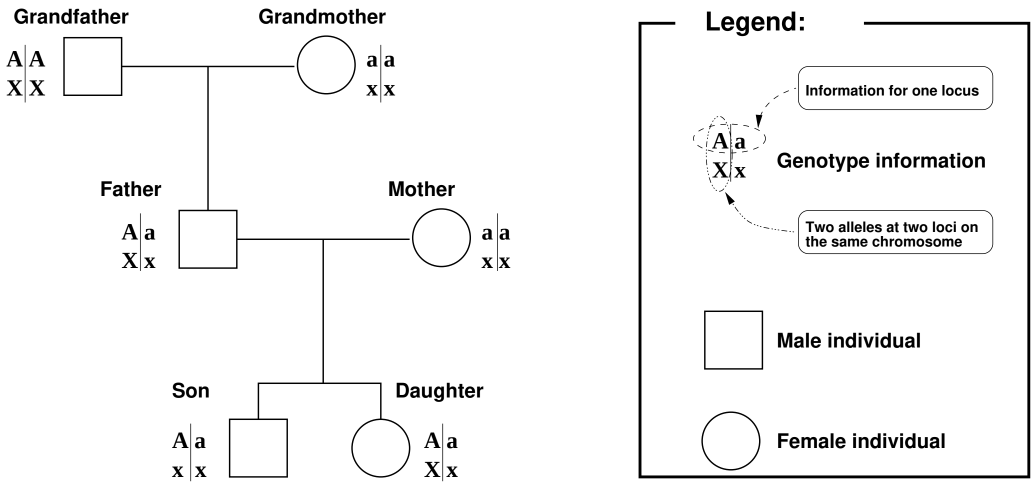

Figure 2-3: An example of a pedigree. The family is constituted by two grandparents, two parents, and two children. Bold letters next to the individuals indicate genotyped information, e.g. the daughter is $\mathbf{A}/\mathbf{a}$ at the first marker and $\mathbf{X}/\mathbf{x}$ at a second marker.

Recombination

Recombination refers to new combinations of genes. This phenomenon arises during meiosis when the two chromosomes of a pair, one inherited from the father and one from the mother, are (most probably) *recombined* to form *four* new unique chromatides, as depicted in Figure 2-2 on the facing page.

The cause of a recombination is the biological event called *crossover*[5]. A crossover involves physical breakage of the chromosome into one paternal and one maternal chromatide, and a joining of the paternal and maternal ends, [SR99, page 40], we have illustrated this in Figure 2-4 on the next page. Note that we write $\mathbf{A}/\mathbf{a}$ when a person has the allelic states $\mathbf{A}$ and $\mathbf{a}$ at a given locus, whereas we write $\frac{\mathbf{A}}{\mathbf{X}}\big|\frac{\mathbf{a}}{\mathbf{x}}$ when $\mathbf{A}$ and $\mathbf{X}$ reside on one chromosome while $\mathbf{a}$ and $\mathbf{x}$ reside on the other. When two alleles at two different loci reside on the same chromosome they are said to be *in phase*, otherwise they are *not in phase*. In the following example we assume to know which alleles reside on which chromosomes, hence the phases of the alleles are assumed to be known. When we do not know the phases of a set of alleles, as it is often the case for linkage analysis, the *phase is unknown*.

Example 2 *In the inheritance example of Figure 2-3 on the facing page the son (denoted Son) is a recombinant, whereas the daughter (denoted Daughter) is a nonrecombinant, with respect to the two loci shown. In the example the son is the recombinant, since the alleles on the father's two chromosomes,* $\frac{\mathbf{A}}{\mathbf{X}}$ *and* $\frac{\mathbf{a}}{\mathbf{x}}$*, have recombined to form then new* $\frac{\mathbf{A}}{\mathbf{x}}$ *combination, so* $\mathbf{A}$ *originates from a different chromosome than* $\mathbf{x}$*.*

Meiosis assures that the number of chromosomes in a human individual is constant, since only one of the four gametes is passed on to a new individual. This gamete is then copied in order to constitute a full chromosome with two sister chromatides for the new individual. Furthermore, meiosis and recombination is the reason that the child's chromosomes, constituting a chromosome pair, are most likely not identical to any of the four chromosome sources of the two parents. Recombinations ensures diversity and uniqueness in a population.

The probability that an odd number of crossovers occur between two loci is termed the *recombination fraction*. Note that if an even number of crossovers (including zero) occur between a pair of loci they are constructed from the same parental chromosome. Thus the individual inheriting this chromosome have inherited the genes at this pair of loci from the same ancestral origin.

Example 3 *Assume that a new chromatide is assembled from one end to the other. Assume that the first part of the chromatide is based on one of the paternal chromatides, a crossover occurs, hence the chromatide is now being constructed from a maternal chromatide. If a new crossover occurs (thus we have had two crossovers so far) the source of the assembly is now back to a paternal chromatide. Multiple switches can occur between the paternal and the maternal chromatides.*

In a sense, recombinations do not occur completely at random, [SR99, page 271]. The concept of *interference* states that in the immediate vicinity of a crossover, it is virtually impossible for another crossover to occur. Thus the recombination fraction remains very close to zero for

[5] Also known as *crossing over*.

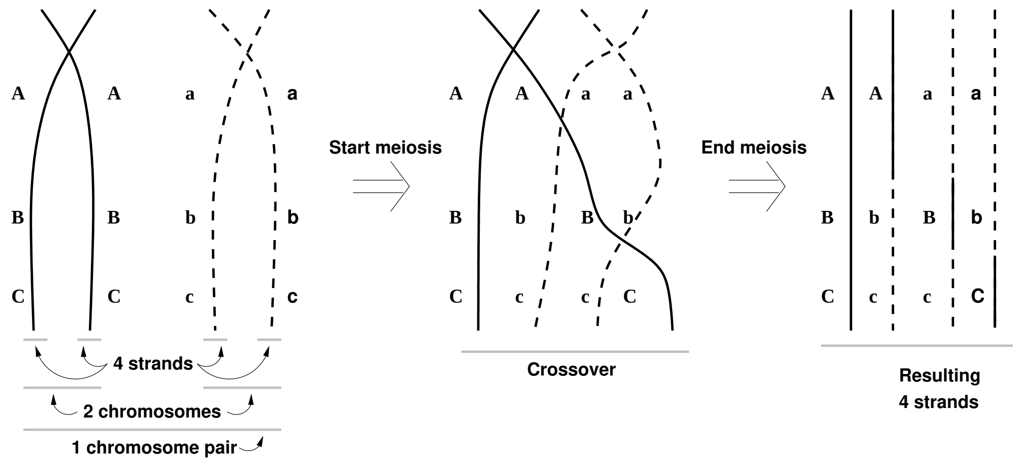

Figure 2-4: The figure illustrates how a crossover might occur on a chromosome, where three marker genes are residing. Hence this meiosis leads to the gametes $\{\mathbf{A},\mathbf{B},\mathbf{C}\}$, $\{\mathbf{A},\mathbf{b},\mathbf{c}\}$, $\{\mathbf{a},\mathbf{B},\mathbf{c}\}$, and $\{\mathbf{a},\mathbf{b},\mathbf{C}\}$, where all but $\{\mathbf{A},\mathbf{B},\mathbf{C}\}$ are recombinations, [Ott99, page 10]. The elements at the starting point is one chromosome pair (two chromosomes), which each consists of two strands (DNA molecules).

some distance along the chromosome near a crossover. The recombination fraction between two loci is related to their physical distance. The mathematical relationship between physical distance and the recombination fraction is given by a *map function*. The most widely used map function is the Haldane map function, [Gud00, page 7]:

$$
\begin{aligned}
d &= -\frac{1}{2} ln(1-2\theta), \\
\theta &= \frac{1}{2}\left(1-e^{-2d}\right),
\end{aligned}
$$

where θ is the recombination fraction and d is the distance in Morgans. On average, approximately one recombination occurs for each 10^8 bps. The distance between two loci is defined as 1 *Morgan* (M), if one recombination can be expected between them during meiosis (hence 1 M is roughly equivalent to 10^8 bps). Thus, there is a correspondence between the distance between two loci and the likelihood of a recombination. The greater the distance, the closer the recombination fraction approaches $\frac{1}{2}$, [Ott99, page 14]. Note that between two loci residing on different chromosome pairs the recombination fraction is $\frac{1}{2}$, as each meiosis on either chromosome are independent. Two loci are said to be *linked* if the recombination fraction between them is less than $\frac{1}{2}$, otherwise the two loci are *unlinked*. This is also an observation made by Mendel, in *Mendel's second law*, he states that unlinked, segregating gene pairs assort independently at meiosis, [Ott99, page 38]. We will refer to Mendel's first and second law as

Mendelian inheritance[6].

Note that the recombination fraction is not constant over the human genome with respect to physical distance. It can deviate along a chromosome and during female meiosis the recombination fraction is on average twice that of male meiosis. In the present study we assume that the recombination fraction is constant, throughout the genome and the same for both males and females[7].

2.5 Locating Trait Genes

The Human Genome project deals with sequencing the chromosomes so that the order of genes is established. However, when this work is completed, the task still remains to uncover which genes are responsible for what trait[8].

Doctors, physicians, psychologists, psychiatrist, etc. might suspect that a trait within their field is caused by some genetic mutation. The suspicion might arise from the fact that the trait is often carried by many in some families and by none in other families. If the trait causing gene(s) could be located, i.e. if the locus or loci could be identified, it is possible to infer the protein(s) that cause a given trait. This knowledge can be used for:

1. **Diagnosis:** If the locus and the allelic state which is responsible for some trait is known, a blood sample would be sufficient to diagnose a person, since the result of a sequencing of the persons DNA at the locus could be matched against the sequence of the trait causing allelic state. It would also be possible to assess the chance of a person developing a trait, by knowing the persons genotype at the trait locus or loci.
2. **Prescribe effective drugs:** Some people respond to some drugs, while others do not. If the responsiveness is dependent on one or more genes, then a determination of a persons allelic state for those genes can lead to the correct drug prescription.
3. **Cure:** If the proteins responsible for a trait are known, it might be possible to develop a drug which surpasses the effect of the proteins or the proteins themselves.

Two kinds of genetic studies exists in the literature. Both aim at locating genes causing traits: *Association studies* and *linkage analysis*, [Cur01]. In association studies the aim is to locate a trait gene by looking for marker alleles more frequent in a group of unrelated individuals carrying a trait, than to a group of people not carrying the trait. E.g. if two groups of 1000 individuals are studied, and 80% of the individuals in one group - the group of people carrying the trait - carry a certain allele, while the same allele is only carried by 30% of individuals in the other group of people not carrying the trait, there is some evidence that the trait and the allele are close to one another (i.e. they are linked). Association studies are only useful when

[6] Often referred to as *the Mendelian laws*.

[7] Gudbjartsson makes the same assumptions in, [Gud00].

[8] Currently, there is quite a large amount of data available, [BO98], however the computational power is insufficient to process this data, [JIS93].

the trait causing gene is located very close to a marker gene, because it is based on a rather sharp division of the two groups (either they have the trait and the marker allele, or they do not have either).

In linkage analysis related people are studied and different pedigrees are studied independently. The aim in linkage analysis is to find the locus of one or more genes causing a trait, by looking at inheritance patterns of markers and comparing these with the inheritance patterns of the trait in question. By analysing how different markers are segregated compared to the set of effected individuals the location of the trait gene can be approximated. Linkage analysis is the focus of the remaining part of this report.

2.6 Summary

In this chapter we have introduced the reader to the biological background of this report, in order to give a basis for understanding the biological context of the rest of this report. We have explained the structure of a DNA molecules, and how they combine to the 23 chromosomes that encodes the genetic information of an individual. Furthermore, we introduced the reader to the process of genetic inheritance. How the alleles are inherited is based on some probability (Mendelian inheritance), which again is influenced by biological properties like recombination fraction, interference, etc. The main task of linkage analysis is to locate the gene on the human genome, which cause the trait of interest.

From now on we will abstract away from biological details. The focus is now on whole genes (markers), which is sufficient to describe linkage analysis formally. Another prime concept is that of pedigrees. As stated earlier, the glossary on page 141 can provide quick reference.

We now concentrate on the task of linkage analysis, thus locating the possible position of a trait causing gene on a single chromosome. The Mendelian laws of inheritance have a major role in linkage analysis. It is safe only to consider a single chromosome pair, since genes on different chromosome pairs assort independently.

Single and Multi Point Analysis

CHAPTER 3

In this chapter we present the formal framework developed in [CHI+02] to describe linkage analysis and extend the framework with respect to multi point linkage analysis. First we introduce the concept of linkage analysis, then we present the formal framework, followed by the definition of single and multipoint point probabilities for inheritance patterns. The algorithms used for single point probability computation were presented and proved in [CHI+02], so we have omitted them here since they are irrelevant for the investigation of heuristics presented in the following chapters. We will, however, go into more detail about the algorithms used for multi point analysis, for which the fastest known are partly based on the fast Fourier transform. For a DD-based data structure[1], such as a MTBDD, BMD, *HDD, etc. to be applicable in linkage analysis it is essential that it is possible to perform *convolution*[2] efficiently, either by Fourier transform or otherwise.

3.1 Introduction to Linkage Analysis

Linkage analysis is a process for constructing *genetic linkage maps*. A genetic linkage map is a description of how a number of loci on a chromosome relate in terms of genetic distance. Hence, a genetic linkage map consists of a number of loci for which the order and relative distances (recombination fractions[3]) are known.

The recent advances in molecular biology have allowed the construction of detailed genetic linkage maps based on molecular *markers*[4](from now on just marker). A marker is a polymorphic DNA or protein sequence derived from a single chromosomal location, [SR99, page 551]. Historically, markers were limited to observable traits known to follow the mendelian[5] inheritance pattern, e.g. blood groups. Today, however, certain highly polymorphic structures of the human DNA can be typed directly on large scale by automated equipment. These structures are called *microsatellites* which are di-, tri-, and tetraneucleotide repeats, and single nucleorite polymorphisms[6].

In the following we give an overview of how linkage analysis is performed, and which data is available.

[1] In Chapter 5 on page 63 we will present a definition of the MTBDDs and investigate the applicability of different DD-based data structures. We refer to Appendix B on page 117 for the definition and examples of some data structures in this family.
[2] We define convolution and show how it is applied in multi point linkage analysis in Section 3.3 on page 29.
[3] See page 9 for more on *recombination fraction*.
[4] See page 6.
[5] See page 8 and 11.
[6] See [SR99, page 273] for a detailed description of the history, use, and typing of markers.

3.1.1 Performing Linkage Analysis

The data needed in order to perform linkage analysis is:

- A genetic linkage map.
- A number of pedigrees.
- Information on the inheritance of markers.
- Trait status for some or all pedigree members.

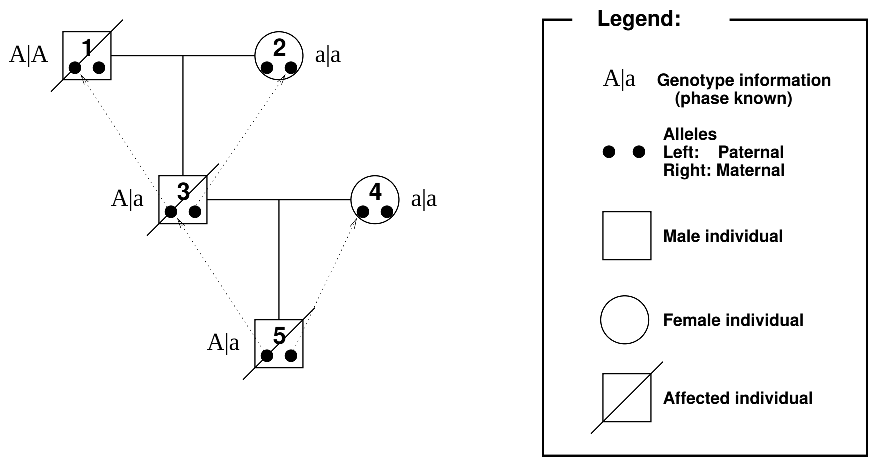

Figure 3-1: A pedigree with trait status and exact information on inheritance of alleles. Note that in reality the phase is often not known.

Figure 3-1 depicts a pedigree with the information for one marker. The information is used to infer evidence that the inheritance pattern of the trait resembles the inheritance pattern of a marker. In Figure 3-1 the inheritance pattern of the trait follows the inheritance of the paternal allele of individual 1. The reason for this could be that the trait causing gene is situated close to the marker, i.e. they are linked which means recombination occurs less frequently than unlinked loci. Unlinked loci are either loci which are far apart on the same chromosome in terms of genetic distance, or loci located on different chromosomes.

Since crossovers occur, it is not always the case that the inheritance pattern of a trait follows exactly that of a marker. For this reason we need a measure of how closely the inheritance patterns of a trait and a marker resembles each other. Such measures are expressed through *scoring functions*. The Likelihood of Odds score (*LOD score*) is one such scoring function. It is out of the scope of this report to define any scoring functions, but for the sake of completeness Appendix A on page 115 gives formal definitions of two commonly used scoring functions, the LOD and Non-Parametric Linkage (NPL) scores.

Linkage analysis is performed for a number of pedigrees on a subset of markers across the genome. If the scoring between the trait and a marker indicates potential linkage, then that region is further analysed by including more markers from the genetic linkage map. Given that a trait causing gene is located in the region under investigation, the new analysis estimates more precisely the location of the trait causing gene.

This process is iterated until no more markers are available in the proximity of the trait causing gene. The genetic linkage map is then updated by introducing a locus representing the trait causing gene.

Using exact knowledge of the inheritance patterns of markers, linkage analysis is straight forward, but in reality many pedigree members are not genotyped making it impossible to infer the precise pattern of inheritance. This and other problems are discussed in the following section.

3.1.2 Missing and Incomplete Data

As stated above, markers where originally limited to simple characteristic traits, such as blood type or eye color. Recent techniques have made it possible to determine the allelic states of microsatellites in human DNA. This method has only been used for a few decades, therefore genotype information is often only available for the youngest generations. Since pedigrees are partly constructed from written documents, such as church records, which contain no information about genetic markers, some individuals in the pedigree might not be genotyped at all. This is often the case for pedigree founders. Furthermore, the phase of the observed alleles for a marker cannot be determined.

Due to these problems, the probabilities for the inheritance patterns at each marker need to be *estimated* based on the information obtained from the genotyped individuals. Since the genotype information is insufficient and the phase is unknown, the precise inheritance pattern becomes impossible to infer. To give an understanding of this issue, we introduce the concepts of *identical/identity by descent*(IBD) and *identical/identity by state*(IBS). Two individuals are IBD if they share inheritance of some founder allele. Two individuals are IBS if they have the same allelic state for some marker, these alleles need not be IBD.

Example 4 (Identity by State/Identity by Descent) *Figure 3-2 is a example of two siblings which are IBS, but not IBD. The individuals 5 and 6 share inheritance of their paternal alleles* (A)*. The allelic states of their maternal alleles* (a) *are identical, but they are not identical by descent because of the pattern of inheritance depicted in the figure.*

Therefore, the precise inheritance pattern is often impossible to infer from the given data, as several inheritance patterns can give rise to the same observed genotype information. However, some of these inheritance patterns might be more likely than others, and thus we need to compute the individual probability of each. These computations are the topic of the rest of this chapter.

Each possible inheritance pattern can then be scored against the inheritance pattern of the trait, using for instance the LOD score. The contribution to the final score of the marker from each inheritance pattern is based on the probability of that pattern.

Computing the probabilities of possible inheritance patterns is performed in two steps, *single point analysis* and *multi point analysis*. These topics are covered in Sections 3.2.2 on page 21 and 3.2.3 on page 23, respectively.

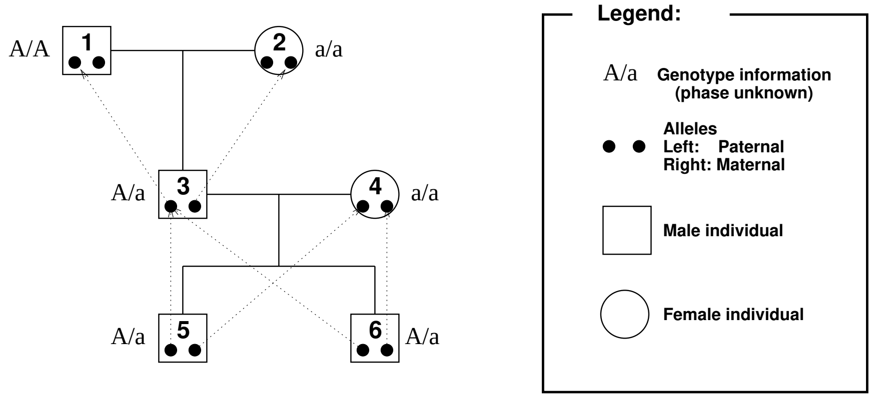

Figure 3-2: IBS vs. IBD. Individuals 5 and 6 are IBS, but since the mother is homozygous they have not necessarily inherited the same maternal allele. In fact in this case they are not IBD.

3.2 Formal Framework

Single point analysis is the process of determining linkage between single markers and traits using only the genotype information available at each marker and the structure of the pedigree. The primary task is computing probability distributions of inheritance patterns at some locus given the available genotype information at that locus. This is called *single point probability computation*.

To be able to express and compute the probability distribution over inheritance patterns, we introduce a formal framework for the biological concepts introduced in the preceding chapter. After the introduction of the formal framework we formally show how the probability distribution is calculated.

3.2.1 Formal Model

The formal model presented in this section represents our earlier work, [CHI$^+$02], but since the definitions of this framework are used throughout the report we include a summarized version.

Pedigree

In Section 2.4 on page 7 the concept of a pedigree was introduced. In the following we formalize this concept.

Definition 1 (Pedigree) *A pedigree is a 4-tuple* $P = \langle F, N, father, mother\rangle$ *with a set of individuals (nodes)* $V = F \cup N$*, where* F *is the founder set and* N *is the non-founder set, and two functions:*

- $father\ :\ N \to V$*, and*
- $mother\ :\ N \to V$*, satisfying the following constraints:*

- *A node cannot be both a founder and a non-founder, that is* $F \cap N = \emptyset$.
- *No node is both a father and a mother, that is:*

$$mother(N) \cap father(N) = \emptyset^7.$$

- *A node is never its own ancestor, that is:*

$$\forall n \in V \;.\; n \notin ancestor(n),$$

where $ancestor : V \rightarrow 2^V$ *is defined recursively as:*

  - *Any non-founder has its father and its mother as ancestors together with the ancestors of both parents, that is:*

$$\begin{aligned}\forall n \in N \;.\; ancestor(n) &= ancestor(father(n)) \;\cup\; ancestor(mother(n)) \\ &\quad \cup \{\; father(n),\; mother(n)\; \}.\end{aligned}$$

  - *Founders have no ancestors, that is:*

$$\forall n \in F \;.\; ancestor(n) = \emptyset.$$

The constraints on the pedigree are introduced to consistently describe the underlying biological model.

Example 5 (A Pedigree) *Figure 3-2 on the facing page is an graphical representation of a pedigree. Formally, the pedigree is defined as follows:*

$$P = \langle F, N, mother, father \rangle, \text{ where:}$$

$$\begin{aligned} F &= \{1, 2, 4\}, \\ N &= \{3, 5\}, \\ mother(3) &= 2, \\ father(3) &= 1, \\ mother(5) &= 4, \text{ and} \\ father(5) &= 3. \end{aligned}$$

Definition 2 (Allele) *The set of alleles in a pedigree* P *is defined as:*

$$\mathsf{A} = \{(n, \varpi) |\; n \in V \text{ and } \varpi \in \{p, m\}\}.$$

[7] Functions over sets are defined the obvious way. E.g. $mother(N) = \{h |\; mother(n) = h \text{ and } n \in N\}$.

We call the subsets $\mathsf{A}_N = \{(n,\varpi)|\ n \in N$ and $\varpi \in \{p,m\}\}$ and $\mathsf{A}_F = \{(n,\varpi)|\ n \in F$ and $\varpi \in \{p,m\}\}$ the *set of non-founder alleles* and the *set of founder alleles*, respectively. (n,p) and (n,m) represent the paternal and maternal alleles of individual n, respectively.

Definition 3 (Marker) *A marker[8] M is defined as a 2-tuple $M = \langle A, \pi \rangle$, where:*

- $A = \{a_1, a_2, \ldots, a_k\}$ *is set of allelic states for the marker M, and*
- $\pi : A \to [0,1]$ *is the allele frequency[9] function, such that:*
  - *The sum of all allele frequencies is 1, that is:*

$$\sum_{a \in A} \pi(a) = 1.$$

The allele frequencies are based on statistical information for a given population.

Notice the difference between the terms *allele* and *allelic state*. An allele is the term used for the section of a chromosome representing a marker. An allele can have a number of states, depending on the allelic states for the marker. Thus, an allele can be thought of as a variable which can be assigned a value, that is, an allele can be assigned an allelic state.

For the remaining part of single point analysis, we assume a fixed underlying pedigree, and we always work with a fixed marker, which we call $M = \langle A, \pi \rangle$. Thus, when we write A or π we refer to allelic states or the allele frequencies for the marker M, unless something else is stated.

Definition 4 (Genotype Information) *Genotype[10] information, $\mathcal{G} = \langle \mathcal{L}, astates \rangle$, for a gene, g, consists of a set of genotyped individuals $\mathcal{L}$ and a function $astates$, where:*

- $\mathcal{L} \subseteq V$, *and*
- $astates\ :\ \mathcal{L} \to \{\{a_1, a_2\} \mid a_1, a_2 \in A\}$.

Notice that the definition indicates that the phase is unknown since there is no reference to maternal and paternal alleles. If an individual is homozygous at a given locus, the $astates$ function returns a set containing just one element.

Example 6 (Genetype information) *The pedigree in Figure 3-2 on page 16 shows genotypic information for all individuals. Thus the set of genotyped individuals $\mathcal{L}$ is equal to the complete node set V. The allelic states of individual 5 are: $astates(5) = \{A, a\}$.*

[8] See page 6.
[9] A more appropriate term would be *allelic state frequency*, however to remain consistent with current practice we call it *allele frequency*.
[10] See Section 2.3 on page 6 for the biological meaning of *genotypes* and *alleles*.

Inheritance Vector

An inheritance vector[11] is a complete specification on how the alleles of founders are inherited by each non-founder. This is a formal way of describing inheritance patterns. We formalize the notion of *inheritance vectors* in the following definition:

Definition 5 (Inheritance Vector) *An inheritance vector v for some pedigree P is a function:*

$$v \ : \ N \times \{p, m\} \to \{p, m\}.$$

The inheritance vector is used to give information about the inheritance of the paternal, p, and maternal, m, alleles for every non-founder (See the example below).

Example 7 (Inheritance Vector) *In the running Example 5 on page 17, an inheritance vector could be:*

$$\begin{aligned} v(3,p) &= p, \\ v(3,m) &= m, \\ v(5,p) &= p, \ \textit{and} \\ v(5,m) &= p. \end{aligned}$$

This would result in the following inheritance pattern (See Figure 3-3):
Individual 3 has inherited his paternal allele from the father of individual 1 and his maternal allele from the mother of individual 2. The two alleles of individual 1 and the two alleles of individual 2 are founder alleles. *Hence, individual 3 has inherited the paternal allele of a founder, individual 1, and the maternal allele of another founder, individual 2. Note that the father of individual 5 (individual 3) does have parents in the pedigree. To determine which founder allele individual 5 has inherited, we use the information that the paternal allele is transmitted from the father's father. This means, that we can find the transmitted founder allele by searching the graph recursively using the inheritance information of individual 3. Thus the paternal allele of individual 5 is the same as the paternal allele of individual 3. Since $v(5, m) = p$ individual 5 has inherited the paternal allele of individual 4, who is a founder.*

Founder Allele Assignment

We need a definition of a *founder allele assignment*, which assigns allelic states to all founder alleles. Such a definition is necessary in order to reason about allelic states of non-founders, since every non-founder has inherited his/her alleles from founders.

Definition 6 (Founder Allele Assignment) *An assignment of allelic states to founder alleles at a marker $M = \langle A, \pi \rangle$, is a function:*

$$Z : F \times \{p, m\} \to A.$$

[11] The notion of *inheritance vectors* was introduced in [LG87] and further used in [KDRDL96].

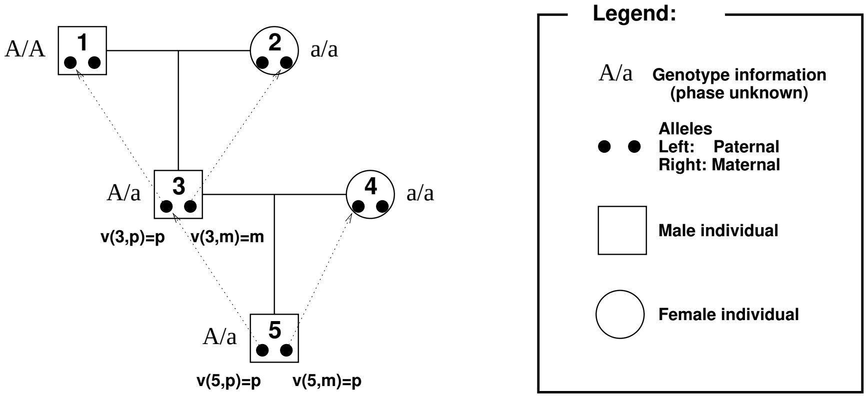

Figure 3-3: Shows how it is determined which founder alleles the non-founders receive given an inheritance pattern at a marker.

The founder allele assignment, Z, assigns allelic states to founders, whereas the inheritance vector, v, only gives information about the path of inheritance of alleles, not the exact allelic state.

Example 8 (Founder Allele Assignment) *Below we show the only possible founder allele assignment for the example:*

$$\begin{aligned}
Z(1,p) &= A, \\
Z(1,m) &= A, \\
Z(2,p) &= a, \\
Z(2,m) &= a, \\
Z(4,p) &= a, \text{ and} \\
Z(4,m) &= a.
\end{aligned}$$

This is because all founders are genotyped.

Assignment of Alleles to Non-founders

From the inheritance pattern, given by an inheritance vector, it is possible to determine which founder alleles a non-founder has inherited. This can be done by the function defined below:

Definition 7 (Inherited Founder Alleles) *Let v be an inheritance vector, then v determines a function:*

$$\mathsf{F}^v : V \times \{p,m\} \to F \times \{p,m\},$$

that assigns to each $n \in V$ a paternal and a maternal founder allele defined recursively:

$$
\begin{aligned}
\mathsf{F}^v(n,p) &= \begin{cases} (n,p) & : \; n \in F \\ \mathsf{F}^v(father(n), v(n,p)) & : \; \textit{otherwise} \end{cases} \\
\mathsf{F}^v(n,m) &= \begin{cases} (n,m) & : \; n \in F \\ \mathsf{F}^v(mother(n), v(n,m)) & : \; \textit{otherwise} \end{cases}
\end{aligned}
$$

Example 9 (Inherited Founder Alleles) *Using the inheritance vector, v, of Example 7 on page 19,* F^v *assigns founder alleles to both non-founders according to:*

$$
\begin{aligned}
\mathsf{F}^v(5,p) &= \mathsf{F}^v(father(5),p) && (\text{since } v(5,p)=p) \\
&= \mathsf{F}^v(3,p) = (father(3),p) && (\text{since } v(3,p)=p) \\
&= (1,p), \\
\mathsf{F}^v(5,m) &= (mother(5),p) = (4,p) && (\text{since } v(5,m)=p), \textit{ and} \\
\mathsf{F}^v(3,m) &= (mother(3),m) = (2,m) && (\text{since } v(3,m)=m).
\end{aligned}
$$

Compatibility

In [KDRDL96] Kruglyak et. al. define the concept *v-compatibility*. This term is an indication of whether the given inheritance vector is compatible with the genotypes observed. We formalize this notion below:

Definition 8 (Compatibility) *Let v be an inheritance vector and $\mathcal{G} = \langle \mathcal{L}, astates \rangle$ some genotype information. Then v is said to be compatible with $\mathcal{G}$ iff there exists a founder allele assignment Z such that the following condition is met:*

$$\forall n \in \mathcal{L} \; . \; \{Z(\mathsf{F}^v(n,p)), Z(\mathsf{F}^v(n,m))\} = astates(n).$$

In this case Z is then said to be a compatible founder allele assignment with respect to v and $\mathcal{G}$.

Example 10 (Compatibility) *The founder allele assignment of Example 8 on the facing page is compatible with the inheritance vector of Example 7 on page 19 and the genotype information of Example 6 on page 18.*

Had the inheritance vector been changed such that $v(5,p) = m$ there would be no compatible founder allele assignment. This is because both individual 2 and 4 are genotyped with only the 'a' allelic state and individual 5 is genotyped with one 'A', which could only have been inherited from individual 1.

3.2.2 Single Point Probability Computation

As mentioned, linkage analysis is the task of determining genetic linkage between a trait and a genetic marker by analyzing inheritance patterns. At hand for the analysis, we have pedigrees with complete or incomplete genotype information[12] and information about the trait for the individuals. In the previous section we saw that not even complete genotype information uniquely determines an inheritance pattern[13]. This means that we cannot determine the true pattern of inheritance in a pedigree. Instead, we calculate the probability of each inheritance pattern given the available genotype information. This task is called *single point probability computation*.

In this section we state the computational task of single point probability computation formally. Given an inheritance vector v and some genotype information $\mathcal{G}$, we are interested in computing $P(v|\mathcal{G})$, the probability of the inheritance vector given the genotype information. Applying Bayes' theorem we find:

$$P(v|\mathcal{G}) = \frac{P(\mathcal{G}|v) \cdot P(v)}{P(\mathcal{G})}, \tag{3-1}$$

where $P(v)$ is the a priori probability of the inheritance vector and $P(\mathcal{G})$ is the probability of the observed genotype information. Since the inheritance vector for a pedigree is not uniquely determined by the genotype information, each inheritance vector has a probability of denoting the true inheritance pattern.

Since we assume Mendelian inheritance the a priori probability for any inheritance vector is the same. Furthermore, the probability of the observed genotype information is independent of the inheritance vector. Thus, $\frac{P(v)}{P(\mathcal{G})}$ of Equation 3-1 is the normalization factor and we deduce that:

$$P(v|\mathcal{G}) \propto P(\mathcal{G}|v), \tag{3-2}$$

for every inheritance vector v.

As can be seen from (3-2) above, the probability of an inheritance vector, v, given some genotype information, $\mathcal{G}$, is proportional to the genotype information given the inheritance vector. We use this result when computing the single point probability distribution.

The probability of some genotype information given an inheritance vector is proportional to the sum of the probabilities of all compatible founder allele assignments. Let v be an inheritance vector, $\mathcal{G}$ some genotype information and $\mathcal{F}_{\mathcal{G}}^{v}$ the set of all compatible founder allele assignments given v and $\mathcal{G}$, that is $\mathcal{F}_{\mathcal{G}}^{v} = \{Z|\ Z$ is a compatible founder allele assignment over $\mathcal{G}$ and $v\}$. Then $P(\mathcal{G}|v)$ is given as:

$$P(\mathcal{G}|v) \propto \sum_{Z \in \mathcal{F}_{\mathcal{G}}^{v}} P(Z). \tag{3-3}$$

The a priori probability on an individual having some allele at any locus is given by the allele frequency π, which is based on statistical population data. Furthermore, the events of two individuals having any two alleles at a given locus are independent. From this we deduce that if Z is a founder allele assignment of a pedigree with founder set, F, then the probability of Z is given as:

$$P(Z) = \prod_{n \in F} \pi(Z(n,p)) \cdot \pi(Z(n,m)).$$

[12]The terms "complete" and "incomplete" genotype information are also referred to as "perfect" and "partial" genotype information, respectively.

[13]The inheritance pattern cannot be uniquely determined since the phase is assumed to be unknown.

In other words, single point probability computation reduces to finding the compatible founder allele assignments for an inheritance vector given some genotype information. Several algorithms for this step exist. For an introduction to two widely used algorithms we refer to [CHI+02] which is based on [Gud00] and [KDRDL96].

3.2.3 Multi Point Probability Computation

The previous section described the computation of the probability distribution of inheritance vectors based on the genotype information available at *a single marker*. Recall from Section 3.1 on page 13 that a genetic linkage map contains multiple markers across each chromosome. Furthermore, Section 2.4 on page 8 states that the probability of recombination between two adjacent markers on a chromosome is given by the *recombination fraction*. For this reason, we can use the knowledge about the probability of inheritance patterns for every marker on a chromosome, to infer something about the probability of inheritance patterns at all other markers on the chromosome. The process of inferring information from all markers to every single marker is called *multi point probability computation*. Hence, in multi point probability computation, the probability distributions over inheritance vectors for every marker on a chromosome are computed with respect to all available genotype information.

Example 11 (Inferring information about one marker to another) *Assume that we have a pedigree with two founders and one non-founder ($child$), and assume that our genetic linkage map contains two markers, namely M_1 and M_2, and the recombination fraction*[14] *between the markers is $\theta_1 = 0.10$. This pedigree has four different inheritance vectors since both the maternal and paternal allele can be inherited either maternally or paternally. If we assume to know the true inheritance vector for M_1: $v_{M_1}(child, p) = p$ and $v_{M_1}(child, m) = p$, we can compute the probability of the four possible inheritance vectors for M_2:*

$v_{M_2}(child, p)$	$v_{M_2}(child, m)$		$P(v_{M_2})$
p	p	$\theta_1^2 =$	*0.81*
p	m	$(1-\theta_1)\theta_1 =$	*0.09*
m	p	$\theta_1(1-\theta_1) =$	*0.09*
m	m	$(1-\theta_1)^2 =$	*0.01*

To understand the calculations behind these numbers, recall that we are assumed to know the true inheritance pattern for M_1 and we know the recombination fraction, which is the probability of a recombination occurring between M_1 and M_2. Since $v_{M_1}(child, p) = p$ and $v_{M_1}(child, m) = p$ the probability of $v_{M_2}(child, p) = p$ and $v_{M_2}(child, m) = p$ is the probability of two recombinations not occurring between M_1 and M_2, hence $(1 - \theta_1)^2 = 0.81$. On the other hand, the probability of $v_{M_2}(child, p) = m$ and $v_{M_2}(child, m) = m$ is the probability of two recombinations occurring, hence $\theta_1^2 = 0.01$. The probability of two remaining inheritance vectors is the probability of one recombination occurring and one not occurring, hence $\theta_1(1 - \theta_1) = 0.09$.

[14] Recall that a genetic linkage map contain markers as well as the recombination fraction between adjacent markers.

In our calculations we have only taken the observed genotype information at marker M_1 into account. If we have some observed genotype information for marker M_2 as well, we perform single point analysis to obtain the single point probability distribution for M_2. The single point probability distribution is multiplied by the distribution shown in table above to obtain the multi point probability distribution over inheritance vectors at M_2, which is now based on the genotype information observed at M_2 as well as M_1.

In the example we assumed that the true inheritance pattern at M_1 was known, of course this is seldom the case. Normally, we would have a probability distribution over inheritance vectors obtained from the single point analysis. The following pages show how to handle the general case, where the true inheritance pattern is not necessarily known.

Multi point analysis is a more accurate than single point analysis since more data is used to calculate the probability distributions over inheritance vectors at every maker.

Formal Model

Before we can define multi point probability computation formally, we need to extend our formal framework with the notion of *recombination fractions*, *Hamming distance*, and *enumeration* of inheritance vectors.

Enumeration of Inheritance Vectors

Assume some enumeration of inheritance vectors,$v_1, v_2, \dots, v_n$, in a pedigree. The probabilities of the different inheritance vectors can then be arranged in a vector of size n where entry i holds the probability of v_i. From now on we assume that the same ordering given in terms of the enumeration is used in a pedigree, regardless of the marker in question.

Example 12 (A Probability Distribution Given as a Vector) *Consider a pedigree with one non-founder, $child$. Assume that we have performed single point analysis and found the inheritance vectors to have the following probabilities: $(p,p) = \frac{1}{10}, (p,m) = \frac{2}{10}, (m,p) = \frac{3}{10}, (m,m) = \frac{4}{10}$, where (m,p) means that the paternal allele has been inherited maternally and the maternal allele has been inherited paternally. Assuming that the order of inheritance vectors is $(p,p),(p,m),(m,p),(m,m)$ we can state the probabilities as a vector P_1:*

$$
\begin{array}{cccc}
 & v(child,p) & v(child,m) & P_1(v) \\
\hline
1 & p & p & \\
2 & p & m & \\
3 & m & p & \\
4 & m & m &
\end{array}
\quad
P_1(v) = \begin{bmatrix} \frac{1}{10} \\ \frac{2}{10} \\ \frac{3}{10} \\ \frac{4}{10} \end{bmatrix}
$$

This example will be further developed through the section.

Definition 9 (Recombination fraction) *The recombination fractions between any two adjacent markers on a chromosome is given by the function θ:*

$$\theta : \mathbb{N}_{m-1} \to \left[0, \frac{1}{2}\right],$$

where $\mathbb{N}_{m-1} = \{1, \ldots, m-1\}$ *is the set of the first* m *natural numbers.* m *is the number of markers on the chromosome.*

We use θ_i to denote $\theta(i)$. Intuitively θ_i denotes the probability of an odd number of crossovers[15] (or recombination) occurring between the i-th and $i+1$-th marker.

We introduce the notion of *Hamming distance* between two inheritance vectors to reason about the number of recombinations that have occurred between two adjacent markers, if one inheritance vector is assumed to describe the correct inheritance pattern at each marker. For any given allele of an individual, the value of the inheritance vector for that allele distinguishes between maternal and paternal inheritance. If two adjacent markers on the same chromosome are inherited differently a recombination has occurred during meiosis. The Hamming distance describes exactly the number of recombinations between two inheritance vectors.

Definition 10 (Hamming Distance) *The Hamming distance between two inheritance vectors* v *and* w *over a pedigree* $P = \langle F, N, father, mother \rangle$ *is defined as:*

$$Ham(v, w) = \sum_{n \in N, \varpi \in \{p, m\}} d_n^{\varpi}; \text{ where}$$

$$d_n^{\varpi} = \begin{cases} 1 \text{ if } v(n, \varpi) \neq w(n, \varpi) \\ 0 \text{ if } v(n, \varpi) = w(n, \varpi). \end{cases}$$

If we have an ordering, and if we write the two inheritance vectors v and w as a strings of p's and m's the Hamming distance between the inheritance vectors is basically the Hamming distance between binary strings. Actually, replacing p with 0 and m with 1 we obtain a binary string representation of v and w. Later we even use the decimal values of the binary strings. We use the term *binary string* even though a string consists of p's and m's, since the meaning of p and m is easier to remember than 0 and 1. Similarly, we also refer to p and m as bits.

Viewing inheritance vectors as binary strings greatly simplifies the enumeration mentioned above. We simply enumerate each inheritance vector according to its integer interpretation, for $p = 0$ and $m = 1$. In the rest of this report we assume this enumeration.

Based on the formal framework, we now describe multi point probability computation.

Let P_i be the vector representation of the single point probabilities for inheritance vectors for marker i under the enumeration just stated, where $P_i(v)$ denotes the probability of v in P_i. The probability of observing one inheritance pattern at one marker when observing another pattern at another marker can then be stated in a transition matrix T_i. Let v and w be inheritance vectors at marker i and $i+1$, respectively, then entry (w, v) (we use the integer interpretation

[15] See page 9 for more on crossover.

of an inheritance vector as a binary string.) of T_i is defined in terms of the recombination fraction and the Hamming distance between v and w:

$$T_i(w,v) = \theta_i^{Ham(w,v)} \cdot (1-\theta_i)^{n-Ham(w,v)}, \tag{3-4}$$

where n is the length of an inheritance vector, when it is written as a binary string, and θ_i is the recombination fraction between marker i and $i+1$. In other words, the entry in the matrix holds the probability of $Ham(w,v)$ recombinations occurring and $n - Ham(w,v)$ not occurring.

Example 13 *Continuing Example 12 we assume that the recombination fraction between the marker described and the following marker is $\theta_1 = \frac{1}{9}$. We can then describe the transition probability between any two inheritance vectors in the transition matrix T_1:*

$$T_1 = \begin{bmatrix} \frac{8}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}\cdot\frac{1}{9} & \frac{1}{9}^2 \\ \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}^2 & \frac{1}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} \\ \frac{8}{9}\cdot\frac{1}{9} & \frac{1}{9}^2 & \frac{8}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} \\ \frac{1}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}^2 \end{bmatrix},$$

under the same ordering as in Example 12.

We define the contribution C_i from marker i to any inheritance vector, w, at marker $i+1$ as a vector where:

$$C_i(w) = \sum_v T_i(w,v) \cdot P_i(v) \tag{3-5}$$

In other words, the contribution is given as matrix by vector multiplication:

$$C_i = T_i \cdot P_i. \tag{3-6}$$

Notice, that this corresponds to the operation performed in Example 11.

Example 14 (Computing a contribution) *In the ongoing example we can find the contribution C_1 from the vector in Example 12 to the next marker on the chromosome using the matrix* *of Example 13:*

$$\begin{aligned}
C_1 = T_1 \cdot P_1 &= \begin{bmatrix} \frac{8}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}\cdot\frac{1}{9} & \frac{1}{9}^2 \\ \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}^2 & \frac{1}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} \\ \frac{8}{9}\cdot\frac{1}{9} & \frac{1}{9}^2 & \frac{8}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} \\ \frac{1}{9}^2 & \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}\cdot\frac{1}{9} & \frac{8}{9}^2 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{10} \\ \frac{2}{10} \\ \frac{3}{10} \\ \frac{4}{10} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{10}\cdot\frac{8}{9}^2 + \frac{2}{10}\cdot\frac{1}{9}\cdot\frac{8}{9} + \frac{3}{10}\cdot\frac{1}{9}\cdot\frac{8}{9} + \frac{4}{10}\cdot\frac{1}{9}^2 \\ \frac{1}{10}\cdot\frac{8}{9}\cdot\frac{1}{9} + \frac{2}{10}\cdot\frac{8}{9}^2 + \frac{3}{10}\cdot\frac{1}{9}^2 + \frac{4}{10}\cdot\frac{1}{9}\cdot\frac{8}{9} \\ \frac{1}{10}\cdot\frac{8}{9}\cdot\frac{1}{9} + \frac{2}{10}\cdot\frac{1}{9}^2 + \frac{3}{10}\cdot\frac{8}{9}^2 + \frac{4}{10}\cdot\frac{1}{9}\cdot\frac{8}{9} \\ \frac{1}{10}\cdot\frac{1}{9}^2 + \frac{2}{10}\cdot\frac{1}{9}\cdot\frac{8}{9} + \frac{3}{10}\cdot\frac{1}{9}\cdot\frac{8}{9} + \frac{4}{10}\cdot\frac{8}{9}^2 \end{bmatrix} \\
&= \begin{bmatrix} \frac{2}{15} \\ \frac{19}{90} \\ \frac{13}{45} \\ \frac{11}{30} \end{bmatrix}
\end{aligned}$$

We have found the contribution to all inheritance vectors at marker $i+1$ from marker i. Let P_{i+1} be the single point probability vector at marker $i+1$. We can then compute the updated probabilities at marker $i+1$ by multiplying the single point probabilities in P_{i+1} with the contribution C_i and normalizing the result to obtain a probability distribution:

$$P_{i+1}^{new} \propto C_i \cdot P_{i+1}.$$

Example 15 *Let:*

$$P_2 = \begin{bmatrix} \frac{2}{3} \\ 0 \\ 0 \\ \frac{1}{3}, \end{bmatrix}$$

be the single point probabilities of inheritance vectors for an adjacent marker of Example 12. We can now calculate the new probability with the contribution in Example 14:

$$P_2^{new} \propto C_1 \cdot P_2 = \begin{bmatrix} \frac{2}{15} \\ \frac{19}{90} \\ \frac{13}{45} \\ \frac{11}{30} \end{bmatrix} \cdot \begin{bmatrix} \frac{2}{3} \\ 0 \\ 0 \\ \frac{1}{3} \end{bmatrix} = \begin{bmatrix} \frac{4}{30} \\ 0 \\ 0 \\ \frac{11}{90} \end{bmatrix},$$

which implies,

$$P_2^{new} = \begin{bmatrix} \frac{12}{23} \\ 0 \\ 0 \\ \frac{11}{23} \end{bmatrix}$$

To calculate the multi point probability for some marker i on a chromosome with m markers, we need to find the contribution from the two adjacent markers. The probability distributions at adjacent markers in turn receive a contribution from their adjacent markers and so forth.

This way any marker gets a contribution from all other markers; the resulting probability is called the *multi point probability*. Imagining that the markers along a chromosome are aligned horizontally, See Figure 3-4. We need to calculate the left and right[16] contributions for each marker i. Let P_i^{sp} be a vector representing the single point probabilities for marker i, we then obtain the *left*, C_i^L, and *right*, C_i^R *contribution vectors* for marker i recursively as:

$$\begin{aligned} \text{base case:} \quad & C_1^L = 1 \\ \text{step:} \quad & C_i^L = T_{i-1} \cdot (P_{i-1}^{sp} \cdot C_{i-1}^L), \end{aligned} \tag{3-7}$$

and similarly for C_i^R:

$$\begin{aligned} \text{base case:} \quad & C_m^R = 1 \\ \text{step:} \quad & C_i^R = T_i \cdot (P_{i+1}^{sp} \cdot C_{i+1}^R). \end{aligned} \tag{3-8}$$

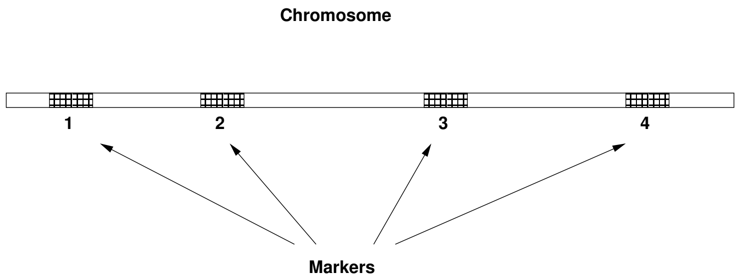

Figure 3-4: A chromosome with four markers orientated horizontally.

Note that we use T_i for the right contribution vector instead of defining a transition probability matrix for the right contribution. This is because the transition probability matrix by definition inherits the property that $T_i(v, w) = T_i(w, v)$ for all inheritance vectors v and w at two adjacent markers, where the entries correspond to the integer interpretations of the inheritance vectors. This implies that the transition matrix for the right contribution would be identical to that of left contribution.

Using the left and right contribution vectors we are now able to calculate the multi point probability, P_i^{mp}, for any marker i:

$$P_i^{mp} \propto C_i^L \cdot P_i^{sp} \cdot C_i^R. \tag{3-9}$$

Example 16 *In the ongoing example consider another marker with single point probabilities:*

$$P_3^{sp} = \begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}.$$

[16] The contribution from all markers located to the left/right of the marker in question.

We want to compute the multi point probabilities for marker 3. From Equation 3-9 we get:

$$P_3^{mp} \propto C_3^L \cdot P_3^{sp} \cdot C_3^R = (T_2 \cdot (P_2^{sp} \cdot C_2^L)) \cdot P_3^{sp} = (T_2 \cdot (P_2^{sp} \cdot (T_1 \cdot P_1^{sp})) \cdot P_3^{sp}.$$

In Example 15 we already calculated $P_2^{sp} \cdot (T_1 \cdot P_1^{sp})$. If we let $\theta_2 = \frac{1}{5}$ be the recombination fraction between marker 2 and 3 we can find the multi point probability:

$$P_3^{mp} \propto \begin{bmatrix} \frac{4}{5}^2 & \frac{4}{5} \cdot \frac{1}{5} & \frac{4}{5} \cdot \frac{1}{5} & \frac{1}{5}^2 \\ \frac{4}{5} \cdot \frac{1}{5} & \frac{4}{5}^2 & \frac{1}{5}^2 & \frac{4}{5} \cdot \frac{1}{5} \\ \frac{4}{5} \cdot \frac{1}{5} & \frac{1}{5}^2 & \frac{4}{5}^2 & \frac{4}{5} \cdot \frac{1}{5} \\ \frac{1}{5}^2 & \frac{4}{5} \cdot \frac{1}{5} & \frac{4}{5} \cdot \frac{1}{5} & \frac{4}{5}^2 \end{bmatrix} \cdot \begin{bmatrix} \frac{4}{30} \\ 0 \\ 0 \\ \frac{11}{90} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix} = \begin{bmatrix} \frac{203}{9000} \\ \frac{23}{2250} \\ \frac{23}{2250} \\ \frac{47}{2250} \end{bmatrix}$$

$$\Updownarrow$$

$$P_3^{mp} = \begin{bmatrix} \frac{203}{575} \\ \frac{4}{25} \\ \frac{4}{25} \\ \frac{188}{575} \end{bmatrix}$$

From Equation 3-9 we see that multi point probability computation is a computationally difficult task in both time and space. We need to represent matrices of dimension $N \times N$ which of course requires $\mathcal{O}(N^2)$ space. This matrix needs to be multiplied with a vector of size N, which is an $\mathcal{O}(N^2)$ operation. For m markers the naive approach has an upper bound of $\mathcal{O}(m \cdot N^2)$. In the following section we shall see how the upper bound running time can be reduced to $\mathcal{O}(m \cdot N \cdot \log N)$ by applying the Fast Fourier Transform (FFT) algorithm.

3.3 Multi Point Analysis using Fourier Transforms

Kruglyak et. al. showed in [KL98] how to use Fourier transforms, [How01, Mor94], to speed-up multi point probability computation. The idea is to view the computation of contribution as convolution. This allows the application of Fourier transform. More specifically, for discrete functions (vectors) an algorithm, fast Fourier transform (FFT), exists which can reduce the complexity of computing left or right contributions from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

3.3.1 Background

Fourier analysis originates from the field of electrical engineering where it is used to transform functions between their representation in the time-domain and the frequency-domain. The terms "time-domain" and "frequency-domain" are taken from the analysis of signals (periodics and pulses) within electrical engineering. In the time-domain a signal can be described by a function $f(t)$, where t is the time and where the value of f is the amplitude of the signal (e.g. a pure tone can be plotted as a sine curve in a 2D coordinate system with time on the x-axis and amplitude on the y-axis). Functions which satisfy some properties[17] can also be represented in

[17] The Fourier integral transforms works for functions in $\mathcal{A}$, the set of piece-wise continuous, absolutely integrable functions on the entire real line, $\mathbb{R}$, [How01, page 273]. The Fourier integral transforms can be extended to

the frequency-domain. The idea of the Fourier transform (or Fourier analysis) is based on what is called *Fourier's bold conjecture*:

Fourier's bold conjecture
Jean Baptiste Joseph Fourier stated that: *Any "reasonable" function can be expressed as a (possibly infinite) linear combination of sines and cosines*, [How01, page 3].

The sequence of coefficients for the sines and cosines are called the *Fourier series* for a function. The Fourier series for a function, $f(t)$, is often given as a function $F(\omega)$, which is f's representation in the frequency-domain. The conversion from the time-domain to the frequency-domain is called *analysis*, while conversion from the frequency-domain to the time-domain is called *synthesis*.

The strength of this approach is that one function has a representation in two different domains. We can choose to analyze or perform operations on the function in whichever domain is better suited since the analysis and the synthesis allow for transformation back and forth. E.g. the *convolution* operation between two functions in the time-domain becomes *multiplication* their representations in the frequency-domain. This operation is of special interest since, as we shall see, the complexity of the convolution operation is an order of magnitude higher than multiplication.

Fourier transforms of discrete functions turn out to be even simpler than for continuous functions. If f is a discrete function over a domain, D, of size $|D|$ then the Fourier transform of f can easily be computed in time $\mathcal{O}(|D|^2)$. However, the fast Fourier transform (FFT) is an algorithm that can reduce this complexity to $\mathcal{O}(|D|\log|D|)$. We mentioned above that convolution in the time-domain becomes multiplication in the frequency-domain. For now, we need only say that convolution between two functions is a $\mathcal{O}(|D|^2)$ operation, but by first performing a Fourier transform on both functions, multiplying their results and performing an inverse Fourier transform on this we have reduced the complexity from $\mathcal{O}(|D|^2)$ to $\mathcal{O}(2\cdot|D|\log|D|+|D|+|D|\log|D|)=\mathcal{O}(|D|\log|D|)$. For large domains this is a significant reduction in complexity.

In the following we explain how the calculation of contributions in multi point probability computation can be seen as convolution of discrete functions.

3.3.2 Application of FFT in Multi Point Probability Computation

Recall from Example 12 that we wrote an inheritance vector as a tuple (p, p), and we could view such tuples as binary strings where p equals 0 and m equals 1. Using this we were able to enumerate inheritance vectors according to their integer interpretation. This notation requires some understanding of which meiosis each index in the binary string refers to. For now, exactly which bits refer which meiosis is not important, however, we will later introduce notation to induce a specific ordering. If there are n meiosis in a pedigree[18] the inheritance vectors as

include functions which are not absolutely integrable, See [How01, chapter 30], however we will not deal with such functions.

[18] There are two meiosis for each non-founder, hence if there are $|N|$ non-founders in a pedigree, the pedigree contains $n = 2|N|$ meiosis.

binary string are elements of $\{p, m\}^n$. Furthermore, as we associate 0 with p and 1 with m an inheritance vector can be represented as an element of $\mathbb{B}^n = \mathbb{Z}_2^n$. $\mathbb{Z}_2$ denotes the set of integers modulo 2, that is $\mathbb{Z}_2 = \{0, 1\}$. $\mathbb{Z}_2^n$ then denotes the direct product of n $\mathbb{Z}_2$, that is $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \cdots \times \mathbb{Z}_2$, n times. As we shall see, this is an important observation for the application of Fourier transforms in multi point linkage analysis.

Let $\oplus, \ominus, \odot$ be the modulo 2 binary operators addition, subtraction, and scalar product, respectively. If $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ are in $\mathbb{Z}_2^n$ we then write:

$$\begin{aligned} x \oplus y &= (x_1 \oplus y_1, \ldots, x_n \oplus y_n) \in \mathbb{Z}_2^n \\ x \ominus y &= (x_1 \ominus y_1, \ldots, x_n \ominus y_n) \in \mathbb{Z}_2^n \\ x \odot y &= (x_1 \odot y_1 \oplus \ldots \oplus x_n \odot y_n) \in \mathbb{Z}_2. \end{aligned}$$

Note that $x \oplus y = x \ominus y$.

By the definition of the transition matrix (3-4) on page 26 we see that each entry depends only on the Hamming distance between the two inheritance vectors. If we are working with a pedigree of $|N|$ individuals we have $n = 2 \cdot |N|$ entries in the inheritance vector. This means that the Hamming distance between two inheritance vectors can at most be n. Consider two inheritance vectors v, w represented as elements of $\mathbb{Z}_2^n$. If we perform bitwise modulo 2 subtraction on v and w the result is also an element of $\mathbb{Z}_2^n$. Whenever v and w have different values for some bit the resulting difference is either 1 or -1 both of which equal 1 modulo 2. If v and w have the same values for some bit the difference is always 0. In other words, the resulting elements have value 1 for all bits where v and w differ, and 0 where they are alike.

Using this we can rewrite the transition matrix T as a vector with entries:

$$T_i(u) = \theta_i^{u_1} \cdot (1 - \theta_i)^{u_0};\ u \in \mathbb{Z}_2^n, \tag{3-10}$$

where u_1 is the number of 1s in u and u_0 is the number of 0s. Applying this definition of the transition "matrix" we can rewrite the contribution vector as follows:

$$C_i(w) = \sum_v T_i(w \ominus v) \cdot P_i(v). \tag{3-11}$$

This operation is known as *convolution*, [How01, Mor94, KL98]. Intuitively, it looks like the operation has the complexity $\mathcal{O}(2^{2n})$, since for every inheritance vector in C_i we sum over all inheritance vector. However, applying fast Fourier transform techniques the running time can be reduced to $\mathcal{O}(n2^n)$. As mentioned previously, the running time of FFT is $=(|D| \log |D|)$, where $|D|$ is the size of the domain, which in our case is $|D| = 2^n$.

Under bitwise addition, $\mathbb{Z}_2^n$ forms a Abelian[19] group for which Fourier transform is especially simple. The following theorem is based on [KL98, JH96, Cai71].

[19] An Abelian group is group for which the group operation is commutative, [All91, Fra94].

Theorem 1 *Any function, f, over the group $\mathbb{Z}_2^n$ under bitwise addition can be decomposed as:*

$$f(w) = \frac{1}{2^n} \sum_v (-1)^{w \odot v} \hat{f}(v), \tag{3-12}$$

with:

$$\hat{f}(v) = \sum_w (-1)^{v \odot w} f(w), \tag{3-13}$$

where $v \odot w$ denotes the bitwise scalar product modulo 2.

For the proof we refer to text books on Fourier transforms, e.g. [How01, Mor94]. In the following we super scribe $\hat{f}$ to refer to the Fourier transform of a function or vectors. Similarly, we super scribe $\overline{f}$ to denote the inverse Fourier transform of a function of vector.

Equation (3-13) is the Fourier transform (or analysis) of f. (3-12) is known as the inverse Fourier transform (or synthesis).

If we view the functions f and $\hat{f}$ of Theorem 1 as vectors and define a matrix:

$$M(v, w) = (-1)^{v \odot w}, \tag{3-14}$$

we are able to describe the Fourier transform of (3-13) as matrix by vector multiplication:

$$\hat{f} = M \cdot f.$$

We then call M the Fourier transform matrix, or just Fourier transform, and we say that $\hat{f}$ is the Fourier transform of f, since f has been transformed into $\hat{f}$.

The Fourier transform on $\mathbb{Z}_2^n$ is also known as the *Walsh-Hadamard transform*, [KL98, Cai71, JH96, Joz98].

We can compute the Fourier transform of the contribution (3-11) as:

Theorem 2 (Convolution on $\mathbb{Z}_2^n$) *Given a probability vector P and a transition matrix T redefined according to (3-10), the convolution vector:*

$$C(w) = \sum_v T(w \ominus v) \cdot P(v), \tag{3-15}$$

has the following Fourier transform:

$$\widehat{C}(u) = \widehat{T}(u) \cdot \widehat{P}(u), \tag{3-16}$$

where $\widehat{T}, \widehat{P}$ are the Fourier transforms of T and P, respectively.

Proof: To prove Theorem 2 we use the proof idea of [Mor94, page 217] and show that the inverse Fourier transform of (3-16) is indeed equal to (3-15). By definition of $\widehat{C}(v)$ and the inverse Fourier transform we arrive at:

$$\overline{\widehat{C}}(w) = \frac{1}{2^n} \sum_v (-1)^{w \odot v} \widehat{C}(v) = \frac{1}{2^n} \sum_v \widehat{T}(v) \widehat{P}(v).$$

Taking the inverse Fourier transform of $\widehat{P}(v)$ we get:

$$\overline{\widehat{C}}(w) = \frac{1}{2^n}\sum_v \Big(\sum_u (-1)^{v\odot u}P(u)\Big)\widehat{T}(v),$$

We then apply the fact that $(-1)^{z\odot x}(-1)^{z\odot y} = (-1)^{z\odot(x\oplus y)}$ for all $x, y, z \in \mathbb{Z}_2^n$ under bitwise modulo 2 addition and scalar product. Furthermore, we interchange the summations and observe that $P(u)$ is independent of v, which gives:

$$\overline{\widehat{C}}(w) = \frac{1}{2^n}\sum_v\sum_u(-1)^{v\odot(u\oplus w)}P(u)\widehat{T}(v) = \sum_u P(u)\frac{1}{2^n}\sum_v(-1)^{v\odot(u\oplus w)}\widehat{T}(v).$$

Finally, we use the definition of inverse Fourier transform and apply that $x\oplus y = x\ominus y$ for all $x, y \in \mathbb{Z}_2^n$:

$$\overline{\widehat{C}}(w) = \sum_u T(w\oplus u)P(u) = \sum_u T(w\ominus u)P(u) = C(w),$$

and the proof is complete, since we have shown that (3-16) is the Fourier transform of the convolution. □

We now show how the fast Fourier transform algorithm achieves an upper bound running time of $\mathcal{O}(|D|\log|D|)$ on a domain of size $|D|$. As mentioned, this gives a running time of $\mathcal{O}(n2^n)$ in the case of probabilities over inheritance vectors. As stated previously, the Fourier transform on $\mathbb{Z}_2^n$ is also known as the Walsh-Hadamard transform which is defined below.

Definition 11 (Walsh-Hadamard Transform Matrix) *The Walsh-Hadamard transform matrix, or just Walsh-Hadamard transform, of order n is defined recursively as:*

$$W_0 = 1; \quad W_n = \begin{bmatrix} W_{n-1} & W_{n-1} \\ W_{n-1} & -W_{n-1}\end{bmatrix}$$

Using the above definition we can restate the Fourier transform of (3-13) and the inverse Fourier transform of (3-12) in terms of matrix by vector multiplication over the Walsh-Hadamard transform:

$$P = \frac{1}{2^n}W_n\cdot\widehat{P}, \tag{3-17}$$

where:

$$\widehat{P} = W_n\cdot P. \tag{3-18}$$

Example 17 (Fourier Transform is Walsh-Hadamard Transform) *This example illustrates that the Fourier transform matrix, (3-14),over $\mathbb{Z}_2^3$ is indeed the Walsh-Hadamard transform matrix. Below, the Fourier transform matrix is depicted and extra braces are inserted to illustrate that it coincides with Definition 11 of the Walsh-Hadamard transform. The instances of $\mathbb{Z}_2^3$ indicated to the left of the matrix indicates the enumeration of the elements in the matrix. The* *columns are enumerated similarly as the rows.*

$$\begin{array}{c}(0,0,0)\\(0,0,1)\\(0,1,0)\\(0,1,1)\\(1,0,0)\\(1,0,1)\\(1,1,0)\\(1,1,1)\end{array}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}+1 & +1\\+1 & -1\end{bmatrix} & \begin{bmatrix}+1 & +1\\+1 & -1\end{bmatrix}\\\begin{bmatrix}+1 & +1\\+1 & -1\end{bmatrix} & \begin{bmatrix}-1 & -1\\-1 & +1\end{bmatrix}\end{bmatrix} & \begin{bmatrix}+1 & +1 & +1 & +1\\+1 & -1 & +1 & -1\\+1 & +1 & -1 & -1\\+1 & -1 & -1 & +1\end{bmatrix}\\\begin{bmatrix}+1 & +1 & +1 & +1\\+1 & -1 & +1 & -1\\+1 & +1 & -1 & -1\\+1 & -1 & -1 & +1\end{bmatrix} & \begin{bmatrix}+1 & +1 & +1 & +1\\+1 & -1 & +1 & -1\\+1 & +1 & -1 & -1\\+1 & -1 & -1 & +1\end{bmatrix}\end{bmatrix}$$

From Definition 11 it is clear that the Walsh-Hadamard transform of order n represents a $2^n \times 2^n$ matrix. Furthermore, the transform on order $n > 0$ can also be defined in terms of the *Kronecker product* of n 2×2 matrices. That is:

$$W_n = \bigotimes_n \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}, \tag{3-19}$$

were $\otimes$ denotes the Kronecker product. We define the Kronecker product in terms of an example.

Example 18 (Kronecker Product) *The Kronecker product can be applied to any two matrices. If A is a $m \times n$ matrix and B is a $p \times q$ matrix then the Kronecker product of A and B is a $(mp) \times (nq)$ matrix illustrated by the following example where A is 2×2 and B is 3×2.*

$$\begin{bmatrix}a_1 & a_2\\a_3 & a_4\end{bmatrix} \otimes \begin{bmatrix}b_1 & b_2\\b_3 & b_4\\b_5 & b_6\end{bmatrix} = \begin{bmatrix}a_1B & a_2B\\a_3B & a_4B\end{bmatrix} = \begin{bmatrix}a_1b_1 & a_1b_2 & a_2b_1 & a_2b_2\\a_1b_3 & a_1b_4 & a_2b_3 & a_2b_4\\a_1b_5 & a_1b_6 & a_2b_5 & a_2b_6\\a_3b_1 & a_3b_2 & a_4b_1 & a_4b_2\\a_3b_3 & a_3b_4 & a_4b_3 & a_4b_4\\a_3b_5 & a_3b_6 & a_4b_5 & a_4b_6\end{bmatrix}$$

In the following theorem we state how the recursive definition of the Walsh-Hadamard transform can be exploited to reduce the running time of the Fourier transform to $\mathcal{O}(|D| \log |D|)$, for any domain of size $|D|$. We only motivate the this algorithm for the Walsh-Hadamard transform, but the general FFT algorithm (See [Mor94, How01]) exploits a similar principle.

The general idea is to define the transform of a probability vector in terms of the transform of two probability vectors of half the size. This idea is then applied recursively. The principle is given in Theorem 3.

First, we need to introduce a notation on probability vectors over inheritance vectors represented as elements of $\mathbb{Z}_2^n$. Let P be a probability vector over inheritance vectors then P_{x_i} is a probability vector of size 2^{n-1} containing only the entries for inheritance vectors with the i-th bit being 1, $(x_i = 1)$. $P_{\overline{x_i}}$ represents the vector containing the inheritance vectors with i-th entry being

0[20], $(x_i = 0)$. We note that P_{x_i} could also be written as $P_{x_i=1}$ (and $P_{\overline{x_i}}$ as $P_{x_i=0}$), however we chose the former notation for brevity. If the variable is understood from the context of not important we use P_0 and P_1 for $P_{\overline{x_i}}$ and P_{x_i}, respectively.

Example 19 (Reduced Probability Vector) *Let P be the following probability vector:*

entry (x_1, x_2, x_3)	P
$(0,0,0)$	0.05
$(0,0,1)$	0.10
$(0,1,0)$	0.15
$(0,1,1)$	0.20
$(1,0,0)$	0.05
$(1,0,1)$	0.10
$(1,1,0)$	0.15
$(1,1,1)$	0.20

For this probability vector we have P_{x_2}, $P_{x_1\overline{x_3}}$ and $P_{x_1x_2\overline{x_3}}$ given as:

$$P_{x_2} = \begin{bmatrix} 0.15 \\ 0.2 \\ 0.15 \\ 0.2 \end{bmatrix}, \quad P_{x_1\overline{x_3}} = \begin{bmatrix} 0.05 \\ 0.15 \end{bmatrix} \text{ and } P_{x_1x_2\overline{x_3}} = \begin{bmatrix} 0.15 \end{bmatrix}$$

The following recursive definition of Walsh-Hadamard transform over $\mathbb{Z}_2^n$ was introduced by Fujita et al. in [FYC+93].

Theorem 3 (Decomposition Theorem for the Walsh-Hadamard Transform) *If P is an ordered probability vector over the inheritance vectors of $\mathbb{Z}_2^n$, such that $P = \left[\begin{smallmatrix} P_0 \\ P_1 \end{smallmatrix}\right]$ where P_0 and P_1 are ordered similarly, etc. then the Fourier transform of P can then be computed recursively as:*

$$\widehat{P} = W_n \cdot P = \begin{bmatrix} W_{n-1}P_0 + W_{n-1}P_1 \\ W_{n-1}P_0 - W_{n-1}P_1 \end{bmatrix} \tag{3-20}$$

Proof: Since we assume that each probability vector P is ordered, such that $P = \left[\begin{smallmatrix} P_0 \\ P_1 \end{smallmatrix}\right]$ it follows immediately from matrix by vector multiplication that:

$$W_n \cdot P = \begin{bmatrix} W_{n-1} & W_{n-1} \\ W_{n-1} & -W_{n-1} \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix} = \begin{bmatrix} W_{n-1}P_0 + W_{n-1}P_1 \\ W_{n-1}P_0 - W_{n-1}P_1 \end{bmatrix}$$

□

[20] For functions, this corresponds to the notion of *positive* and *negative cofactors* as we introduce in Section 5.1.1.

The recursive structure of Theorem 3 is an obvious "divide and conquer" approach which for inheritance vectors over $\mathbb{Z}_2^n$ has an upper bound running time of $\mathcal{O}(2^n \log_2 2^n) = \mathcal{O}(n2^n)$. For any $0 < i \leq n$ the operation $W_i P^i$[21] requires the summation and subtraction of two vectors of size 2^{i-1}. In other words, $W_i P^i$ is an $\mathcal{O}(2^i)$ operation. Figure 3-5 depicts how performing $W_n P^n$ requires two calls of type $W_{n-1} P^{n-1}$, each of which requires two calls of type $W_{n-2} P^{n-2}$, etc. This continues until we have 2^n operations of type $W_0 P^0$. Summing all operations at a single layer reveals that at layer i we always have 2^i operations of complexity $\mathcal{O}(2^{n-i})$, that is, the complexity of all operations at any layer i is $\mathcal{O}(2^n)$. Since there are $\log_2 2^n = n$ layers of the binary tree in Figure 3-5, the total complexity of $W_n P^n$ is $\mathcal{O}(n2^n)$.

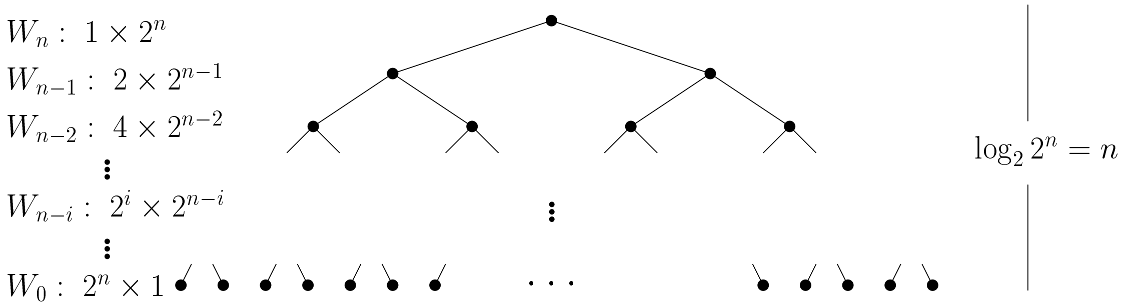

Figure 3-5: The recursive nature of the Walsh-Hadamard transform. The transform of a vector of size 2^n is computed from 2^n vectors of size 1, which are turned into 2^{n-1} vectors of size 2, and so forth until we create the vector of size 2^n from 2 vectors of size 2^{n-1}.

The beauty of this approach is that we only construct the Walsh-Hadamard transform for W_0, which is the simple $[1]$ "matrix". That is, we can construct the Walsh-Hadamard transform of any vector of size 2^n by taking 2^n vectors of size 1 and turning them into 2^{n-1} vectors of size 2, which in turn are turned into 2^{n-2} vectors of size 4, etc. This continues until we construct the Walsh-Hadamard transform of the vector of size 2^n from 2 vectors of size 2^{n-1}. By reading Figure 3-5 in a bottom-up fashion it is illustrated how the bottom row uses W_0, which is used to perform W_1, etc.

Furthermore, the Walsh-Hadamard transform of any vector can be performed in place, [PSAT92], that is, the we can compute the transform by storing all intermediate results in the vector.

The running time of multi point probability computation using (3-20) is, $\mathcal{O}(n2^n)$, which is an order of magnitude better than the naive approach. This running time is achieved by first performing a Fourier transform of the probability vector P and the transition matrix T, multiplying the two and then performing an inverse Fourier transform to obtain the final contribution C. In other words, the computation of the contribution requires three Fourier transforms and one multiplication.

This can actually be reduced to only two Fourier transforms by using the observation made in [KL98] that the Fourier transform of T is an especially simple vector. We summarize the result in the following theorem:

[21] For clarity we super scribe matrices by an i to indicate that it has 2^i entries.

Theorem 4 (The Fourier transform of T) *Let T be the transition matrix rewritten as a vector according to (3-10):*

$$T(u) = \theta^{u_1} \cdot (1-\theta)^{u_0};\ u \in \mathbb{Z}_2^n,$$

where θ is the recombination fraction and u_1 is the number of 1s in u and u_0 is the number of 0s.

The Fourier transform of T is given by:

$$\widehat{T}(u) = (1-2\theta)^{u_1}.$$

We refer to [KL98] for the proof.

Theorem 3 states how the Walsh-Hadamard transform of a probability vector can be computed recursively. The methods requires an ordering of the elements in the probability vector that respects $P = \left[\begin{smallmatrix} P_{\overline{x_i}} \\ P_{x_i} \end{smallmatrix}\right]$ whenever we decomposed P with regard to x_i. This is easily achieved by using the *natural order*[22]. Let P be a probability vector over $\mathbb{Z}_2^n$ the natural order is then given as:

Vector Entry	Inheritance vector $(x_1, \ldots, x_n)$
0	$(0, \ldots, 0, 0, 0)$
1	$(0, \ldots, 0, 0, 1)$
2	$(0, \ldots, 0, 1, 0)$
3	$(0, \ldots, 0, 1, 1)$
$\vdots$	$\vdots$
$2^n - 1$	$(1, \ldots, 1, 1, 1)$

In other words, the entry of any element in $\mathbb{Z}_2^n$ corresponds to its integer interpretation as a binary string. For instance, the probability of the inheritance vector $(1, 0, 1)$ has entry 5, counting from 0 to 7. In the rest of this report we refer only to the natural order, which is identical to the enumeration induced by the integer interpretation.

Using the natural order of inheritance vectors in the probability vector, computing the recursive Walsh-Hadamard transform is simple: First decompose with respect x_1 then x_2 etc. This way we are guaranteed the ordering of the probability vector required by Theorem 3.

3.4 Summary

In this chapter we formalized the biological concepts introduced in Chapter 2. The formalization was a summary of our previous work, [CHI+02].

The formal framework enabled us to define single and multi point probability computation. We have explained how multi point probability computation is performed in practice using Fourier transform. We also showed how the application of Fourier transforms reduced the complexity of multi point probability computation from $\mathcal{O}(2^{2n})$ to $\mathcal{O}(n2^n)$.

[22] Note that we used the natural order to prove the decomposition theorem of the Walsh-Hadamard transform.

A Faster Algorithm

Chapter 4

We have seen in the previous chapter how multi point probability computation can be speeded-up by applying Fourier transforms. The optimized algorithm requires just two Fourier transforms and a multiplication of two vectors to compute the contribution from a marker. This method is currently applied in two of the more known linkage analysis programs, Allegro and GeneHunter, [Gud00, KL98].

In this chapter we propose a new method, which offer no improvement in terms of theoretical complexity, however, multi point probabilities are computed using only a single transform and no multiplication.

First, we introduce a new transform called *recombination transform* and show its correctness in terms of complexity and computation of multi point probabilities. Second, we extend the transform to accommodate for *founder reduction*, which is way of reducing the inheritance vector space by a factor two for every founder in the pedigree. Founder reduction is used in both Allegro and GeneHunter.

4.1 Recombination Transform

The basic idea is to use a transform with similar recursive decomposition as the Walsh-Hadamard transform. We name this transform the *recombination transform* since its definition depends on the recombination fraction between two markers.

In [MDK01] Kruglyak et al. defines a similar transform to the recombination transform. However, the article still recommends the fast Fourier method for multi point linkage analysis, since the recombination transform is not directly applicable to *founder reduced* inheritance vector spaces. First, we introduce the recombination transform in terms of *complete* inheritance vectors, e.g. inheritance vectors which are not founder reduced. Second, we define founder reduction on inheritance vectors. Finally, we propose a extension of the recombination transform for founder reduced inheritance vectors.

4.1.1 Recombination Transform on Complete Inheritance Vectors

Below we define the recombination transform for multi point probability computation and prove that it shares the recursive structure of the Walsh-Hadamard transform.

Definition 12 (Recombination Transform) *Let θ be the recombination fraction between two markers. The recombination transform of order n over θ, $\mathcal{R}_n^\theta$, is defined recursively as:*

$$\mathcal{R}_1^\theta = \begin{bmatrix} (1-\theta) & \theta \\ \theta & (1-\theta) \end{bmatrix} \quad \mathcal{R}_n^\theta = \begin{bmatrix} (1-\theta)\cdot\mathcal{R}_{n-1}^\theta & \theta\cdot\mathcal{R}_{n-1}^\theta \\ \theta\cdot\mathcal{R}_{n-1}^\theta & (1-\theta)\cdot\mathcal{R}_{n-1}^\theta \end{bmatrix}.$$

It follows immediately that the recombination transform of order n can be written as the following Kronecker product:

$$\mathcal{R}_n^\theta = \bigotimes_n \begin{bmatrix} (1-\theta) & \theta \\ \theta & (1-\theta) \end{bmatrix}. \tag{4-1}$$

In Section 3.2.3 on page 23 we defined the transition matrix between marker i and $i+1$ with recombination fraction θ_i as:

$$T_i(w,v) = \theta_i^{Ham(w,v)} \cdot (1-\theta_i)^{n-Ham(w,v)},$$

where $Ham(w,v)$ denotes the Hamming distance between two inheritance vectors. If we view inheritance vectors as elements of $\mathbb{Z}_2^n$ and order the transition matrix according to the natural order[1], we observe that T_i is identical to $\mathcal{R}_n^{\theta_i}$. Using the naturally ordered probability vector, P_i, over inheritance vectors we can compute the contribution as:

$$C_i(w) = \sum_v T_i(w,v)\cdot P_i(v)$$

$$\Updownarrow$$

$$C_i = T_i \cdot P_i = \mathcal{R}_n^{\theta_i}\cdot P_i. \tag{4-2}$$

Since the recombination transform of a probability vector is so closely related to the contribution, we denote the recombination transform of a probability vector, P, as C. Similarly to the Walsh-Hadamard transform we are now able to recursively compute the contribution:

Theorem 5 (Decomposition Theorem for the Recombination Transform) *If P is a naturally ordered probability vector over inheritance vectors of $\mathbb{Z}_2^n$ then the recombination transform over θ of P can then be computed recursively as:*

$$\mathcal{R}_n^\theta \cdot P = \begin{bmatrix} (1-\theta)\cdot\mathcal{R}_{n-1}^\theta\cdot P_0 + \theta\cdot\mathcal{R}_{n-1}^\theta\cdot P_1 \\ \theta\cdot\mathcal{R}_{n-1}^\theta\cdot P_0 + (1-\theta)\cdot\mathcal{R}_{n-1}^\theta\cdot P_1 \end{bmatrix} \tag{4-3}$$

Proof: Since we assume that each probability vector P is naturally ordered, that is $P = \left[\begin{smallmatrix} P_0 \\ P_1 \end{smallmatrix}\right]$ it follows immediately matrix by vector multiplication that:

$$\begin{bmatrix} (1-\theta)\cdot\mathcal{R}_{n-1}^\theta & \theta\cdot\mathcal{R}_{n-1}^\theta \\ \theta\cdot\mathcal{R}_{n-1}^\theta & (1-\theta)\cdot\mathcal{R}_{n-1}^\theta \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \end{bmatrix} = \begin{bmatrix} (1-\theta)\cdot\mathcal{R}_{n-1}^\theta\cdot P_0 + \theta\cdot\mathcal{R}_{n-1}^\theta\cdot P_1 \\ \theta\cdot\mathcal{R}_{n-1}^\theta\cdot P_0 + (1-\theta)\cdot\mathcal{R}_{n-1}^\theta\cdot P_1 \end{bmatrix}$$

□

[1] See page 37 for the definition of the natural order.

Obviously, the decomposition of the recombination transform has the same upper bound running time as that of the recursive Walsh-Hadamard transform, which is $\mathcal{O}(n2^n)$ (for inheritance vectors over $\mathbb{Z}_2^n$). The reduction in running time is achieved since we only need to perform the recombination transform once to compute the contribution from one marker to another, whereas the Fourier transform approach requires two transforms and one multiplication.

Using this approach we can reduce the computation of the multi point step from two Fourier transforms and one multiplication to a single recombination transform with equal complexity as that of the Fourier transform.

4.2 Founder Reduced Recombination Transform

In 1996, Kruglyak et al. introduced the notion of *founder reduction*, [KDRDL96, Gud00]. Founder reduction is a way of optimizing the computational task of single and multi point linkage analysis by defining equivalence classes on inheritance vectors. Kruglyak et al. observed that for every founder in a pedigree computations can be reduced by a factor of two by only considering half the set of inheritance vectors.

The idea is based on the fact that parents of founders are unavailable for study. Thus, the probability of an inheritance vector given some genotype information is unaffected if we consistently switch all bits with value 0 (p) to 1 (m) and all bits valued 1 to 0 for all meiosis concerning a founder. E.g. the meiosis concerning the founder $father$ in Figure 4-1 are the paternal alleles of son and $daughter$. By performing the switch, the genotype restrictions induced on the maternal allele of the founder is changed to the paternal allele and vice versa. Obviously, this has no impact on the calculation of single point probabilities[2] for the two inheritance vectors. For this reason we need only consider half the set of inheritance vectors for further study. This can be applied for all founders in a pedigree achieving a reduction of computation by a factor of 2^f where f is the number of founders.

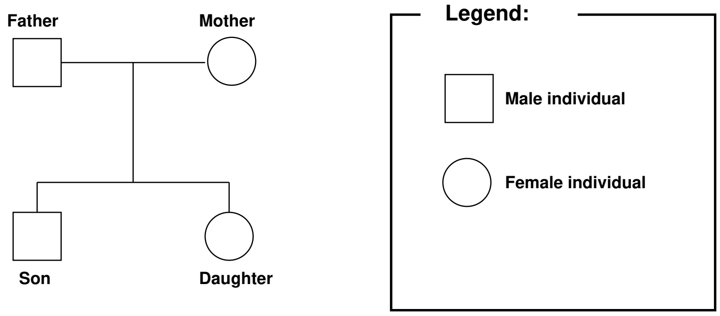

Figure 4-1: A simple pedigree with a complete inheritance vector of length 4 and founder reduced inheritance vector of length 2.

[2] See Section 3.2.2 on page 21 for more on the single point probability computation.

Example 20 (Switching Bits for Meiosis Concerning Founders) *Given some inheritance vector v for the pedigree in Figure 4-1, $v(son, p) = m, v(daughter, p) = m, v(son, m) = p$, and $v(daughter, m) = m$ (the element $(1, 1, 0, 1)$ in $\mathbb{Z}_2^4$). The inheritance vector for which all bits have been switched for meiosis concerning the founder $father$ is w, $w(son, p) = p, w(daughter, p) = p, w(son, m) = p$, and $w(daughter, m) = m$ (the element $(0, 0, 0, 1)$ in $\mathbb{Z}_2^4$). For any founder allele assignment, Z^v, compatible with v there is a founder allele assignment, Z^w, compatible with w, for which Z^v and Z^w are identical for all founders except $father$, where $Z^v(father, m) = Z^w(father, p)$ and $Z^v(father, p) = Z^w(father, m)$. Hence, switching all bits concerning meiosis for some founder does not change the probability of an inheritance vector.*

For the theoretical complexity of single and multi point analysis founder reduction has no impact, since computations still grow exponentially in the number of non-founders in the pedigree. However, in practice this produces a significant reduction in space and time requirements, [KDRDL96].

Variable Ordering

In this section it becomes important which meiosis the individual bits represent, and for this reason we define the ordering of an the elements in an inheritance vector in terms on an *inheritance vector layout*.

Definition 13 (Inheritance Vector Layout) *An inheritance vector layout is an ordered list of pairs of the type $N \times \{p, m\}$, where N is the set of non-founders in a pedigree. Every possible pair is present in the list with no duplicates.*

Since each element $N \times \{p, m\}$ is present in the ordered list we can associate with an inheritance vector layout a bijection, o, of $N \times \{p, m\}$ onto $\{1, \dots, 2 \cdot |N|\}$. Intuitively, given some $(child, p)$, $o(child, p)$ return the index of the paternal meiosis of $child$ in the inheritance vector. We will even refer to o as the inheritance vector layout, since each such layout has a unique bijection, o. Since o is a bijection we define its inverse, o^{-1}, from $\{1, \dots, 2 \cdot |N|\}$ onto $N \times \{p, m\}$ in the obvious way.

Founder Bit Mask

To introduce the recombination transform on founder reduced probability vectors we need to know which bits in an inheritance vector concern meiosis of founders. For this purpose we associate with each founder in the pedigree a *founder bit mask*.

Definition 14 (Founder Bit Mask) *Given some inheritance vector layout, $o : N \times \{p, m\} \to \{1, \dots, 2 \cdot |N|\}$, then the founder bit mask $s_h \in \mathbb{Z}_2^n$ for some founder h is given as:*

$$s_h(i) = \begin{cases} 1 : o^{-1}(i) \text{ is a meiosis of } h \\ 0 : \text{ otherwise,} \end{cases}$$

where $s_h(i)$ denotes the value of the i-th element of s_h.

Intuitively, s_h has bit value 1 for all meiosis concerning h and bit value 0 for the rest.

Example 21 (Founder Bit Mask) *Let the complete inheritance vector* (x_1, x_2, x_3, x_4) *for the pedigree in Figure 4-1 have the inheritance vector layout, o, such that x_1 corresponds to* (son, p) $(o(son, p) = 1)$*,* x_2 *to* $(daughter, p)$ $(o(daughter, p) = 2)$*,* x_3 *to* (son, m) $(o(son, m) = 3)$*, and* x_4 *to* $(daughter, m)$ $(o(daughter, m) = 4)$*. The bit mask of $father$, which is a bit mask for the meiosis concerning $father$, is then* $(1, 1, 0, 0)$*, and for $mother$ it is* $(0, 0, 1, 1)$*.*

We apply the formalism of founder reduction stated in the original paper of Kruglyak et al., [KDRDL96], in which inheritance vectors are seen as elements of $\mathbb{Z}_2^n$. We observe, that switching bits representing meiosis concerning some founder, h corresponds to adding the bit mask, s_h, of h. Using this, we are now able to state the *founder reduction theorem*. Example 22 defines the notion of cosets on groups which is used in the founder reduction theorem.

Example 22 (Coset) *Given a group* $\langle G, +\rangle$*, where '+' is the group operation, and a subgroup H of G, then the left coset of H with respect to some $g \in G$ is defined as* $g + H = \{g + h \mid h \in H\}$*. Similarly, the right coset of H with respect to some $g \in G$ is defined as* $H + g = \{h + g \mid h \in H\}$*. By definition left and right cosets are identical for Abelian groups, since the group operation is commutative which implies that* $g + h = h + g$*.*

Theorem 6 (Founder Reduction) *For a pedigree with f founders, let* $\{s_1, \ldots, s_f\}$ *be the set of founder bit masks. Let S be the subgroup of $\mathbb{Z}_2^n$ generated by* $\{s_1, \ldots, s_f\}$[3]*. For any inheritance vector, v, the equivalence class,* $[\![v]\!]_S$*, of inheritance vectors which differ only by consistently changing all bits for some founders is precisely the coset of S with respect to v*[4]*. Then for each equivalence class any two elements have identical single, P^{sp}, and multi point, P^{mp}, probabilities, that is:*

$$\forall u \in [\![v]\!]_S \; . \; P^{sp}(u) = P^{sp}(v) \text{ and } P^{mp}(u) = P^{mp}(v).$$

Intuitively, S contains all elements which can be generated by bitwise summation of a number of founder bit masks. Since for each element in $t \in S$, any founder bit mask has either been one of the elements generating t or not, there are exactly 2^f elements in S.

The equivalence class, $[\![v]\!]_S$, of any inheritance vector, v, consists of v itself and any inheritance vector which is identical to v except for switching bits for a number of founders. E.g for the inheritance vector $(1, 1, 0, 1)$[5] of the pedigree in Figure 4-1, the equivalence class is given as $[\![(1, 1, 0, 1)]\!]_S = \{(1, 1, 0, 1), (0, 0, 0, 1), (1, 1, 1, 0), (0, 0, 1, 0)\}$. That is, the element itself, $(1, 1, 0, 1)$, the element added with the bit mask of $father$, $(0, 0, 0, 1)$, the element added with

[3] This is equivalent to defining S as the subspace spanned by $\{s_1, \ldots, s_f\}$, that is $S = Span\{s_1, \ldots, s_n\}$, where we use the notation of [Lay97].

[4] Hence, the equivalence classes $[\![v]\!]_S$ are the quotient groups $\mathbb{Z}_2^n/S$ of $\mathbb{Z}_2^n$. However, since the inheritance vectors in a quotient group are identical with respect to single and multi point probabilities we prefer the notion of equivalence class.

[5] This the inheritance vector v from Example 20.

the bit mask of $mother$, $(1,1,1,0)$, and finally the element added with both the bit masks of $father$ and $mother$, $(0,0,1,0)$. Since we can combine switches of bits concerning meiosis for founder in 2^f ways, the equivalence class of an inheritance vector contains 2^f elements.

We refer the keen reader to our earlier work, [CHI$^+$02], where we prove that switching bits concerning meiosis of founders in inheritance vectors has no impact on single and multi point probabilities, hence that equivalence classes are preserved in multi point probability computation.

Example 23 (Founder Reduction) *If i, j, and k ($1 \le i < j < k \le n$) denote the bits in an inheritance vector concerning some founder, then Theorem 6 states that for the single point probability vector, P^{sp}, and the multi point probability vector, P^{mp}, over $\mathbb{Z}_2^n$ we have:*

$$\begin{aligned} P^{sp}(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_k,\ldots,x_n) &= P^{sp}(x_1,\ldots,\overline{x}_i,\ldots,\overline{x}_j,\ldots,\overline{x}_k,\ldots,x_n) \\ P^{mp}(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_k,\ldots,x_n) &= P^{mp}(x_1,\ldots,\overline{x}_i,\ldots,\overline{x}_j,\ldots,\overline{x}_k,\ldots,x_n), \end{aligned}$$

for any inheritance vector $(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_k,\ldots,x_n)$.

Due to founder reduction we need only focus on a reduced set of inheritance vectors, that is, we consider only equivalence classes over inheritance vectors. For this purpose we define a *founder reduced probability vector* over $\mathbb{Z}_2^{n-f}$ where each element represents one of the 2^{n-f} equivalence classes. The probability represented in a founder reduced probability vector is the sum of the probabilities of the elements in the equivalence class. Since all probabilities are equal within an equivalence class, the summed probability is just 2^f times the probability of any element in the equivalence class.

The founder reduction theorem states that probabilities in equivalence classes are the same after single point as well as after multi point probability computation. This is because any two inheritance vectors in the same equivalence class, w and w', at some marker get the same contribution from an adjacent marker, that is:

$$C(w) = \sum_v T(w,v) \cdot P(v) = \sum_v T(w',v) \cdot P(v) = C(w').$$

This holds since any two inheritance vectors, w and w', in the same equivalence class get the same contribution from any other equivalence class, $[\![v]\!]_S$:

$$\forall w, w' \in [\![w]\!]_S \;.\; \sum_{u \in [\![v]\!]_S} T(w,u) = \sum_{u \in [\![v]\!]_S} T(w',u). \tag{4-4}$$

The probabilities, $P(u)$ and $P(u')$, of any two $u, u' \in [\![v]\!]_S$ are equal and have been left out of (4-4).

The transition probabilities of (4-4) are actually pairwise identical, meaning that for any transition probability $T(w,u)$ there is a transition probability $T(w',u')$ with equal probability. This is depicted in Figure 4-2. Stated formally:

$$\forall w, w' \in [\![w]\!]_S \;.\; \exists u, u' \in [\![v]\!]_S \;.\; T(w,u) = T(w',u'). \tag{4-5}$$

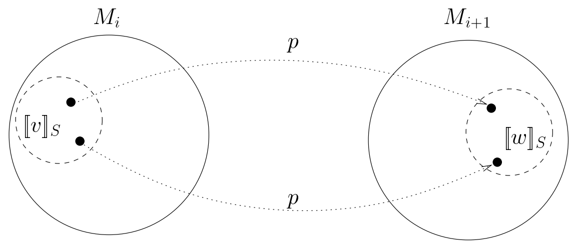

Figure 4-2: For any two equivalence classes $[\![v]\!]_S$ and $[\![w]\!]_S$, if the is a transition $T(v', w') = p$ for any $v' \in [\![v]\!]_S$ and $w' \in [\![w]\!]_S$, then there is also a transition $T(v'', w'') = p$ for some $v'' \in [\![v]\!]_S, w'' \in [\![w]\!]_S$.

These pairwise identical transition probabilities are easily identified. Take any two inheritance vectors in an equivalence class $w, w' \in [\![v]\!]_S$, if we know that $T(w, u) = p$ for some $u \in [\![v']\!]_S$, we would like to identify the element $u' \in [\![v']\!]_S$ such that $T(w', u') = p$. We know that for some element, $t \in S$[6], $w' = w + t$ and the element u' is simply given as $u' = u + t$. We illustrate this in the following example.

Example 24 (Pairwise Identical Probabilities) *Consider a pedigree with only one founder[7] for which the i-th, $i+1$-th, and $i+2$-th entries in the inheritance vector represent the three meiosis for the founder. The bit mask, s, for the founder is given as:*

$$s = (\overbrace{0, \ldots, 0}^{i-1}, 1, 1, 1, \overbrace{0, \ldots, 0}^{n-(i+2)}),$$

where n is the last index. The subgroup, S, of $\mathbb{Z}_2^n$ generated by $\{s\}$ is:

$$S = \{(0, \ldots, 0), s\}.$$

For any inheritance vector $x = (x_1, \ldots, x_n)$, the equivalence class of x is:

$$[\![x]\!]_S = \{(x_1, \ldots, x_i, x_{i+1}, x_{i+2}, \ldots, x_n), (x_1, \ldots, \overline{x}_i, \overline{x}_{i+1}, \overline{x}_{i+2}, \ldots, x_n)\}.$$

We call the latter element $\overline{x} = x + s$. Now take any other inheritance vector and its equivalence class $[\![y]\!]_S$ consisting of y itself and the vector for which the i-th, $i+1$-th, and $i+2$-th bits are switched, called $\overline{y} = y + s$. We now observe that the Hamming distance from x and $\overline{x}$ to y are the same except for the three bits concerning the founder. Actually, since $\overline{x}$ have the exact opposite entries of x for these three bits we have:

$$Ham((x_i, x_{i+1}, x_{i+2}), (y_i, y_{i+1}, y_{i+2})) = Ham((\overline{x}_i, \overline{x}_{i+1}, \overline{x}_{i+2}), (\overline{y}_i, \overline{y}_{i+1}, \overline{y}_{i+2})),$$

[6] S is the subgroup of $\mathbb{Z}_2^n$ generated by the set of founder bit masks. See the founder reduction theorem (Theorem 6 on page 43) for the introduction of S.

[7] Obviously, this is not possible in actual pedigrees, but it serves as exemplification.

and,

$$Ham((x_i, x_{i+1}, x_{i+2}), (\overline{y}_i, \overline{y}_{i+1}, \overline{y}_{i+2})) = Ham((\overline{x}_i, \overline{x}_{i+1}, \overline{x}_{i+2}), (y_i, y_{i+1}, y_{i+2})).$$

This implies that $Ham(x, y) = Ham(\overline{x}, \overline{y})$ *and* $Ham(x, \overline{y}) = Ham(\overline{x}, y)$ *and we arrive at* $T(x, y) = T(\overline{x}, \overline{y})$ *and* $T(x, \overline{y}) = T(\overline{x}, y)$ *which is exactly what (4-5) states.*

The complete proof of Equation (4-5) follows the same line of reasoning as the preceding example.

The following layout, o, of inheritance vectors is assumed for convenience in the remaining part of this section. For a pedigree with f founders we assume the complete inheritance vector to be ordered such that the last f bits of the inheritance vector represents one meiosis from each founder. That is, for each founder, h, there is a meiosis (v, p) where $father(v) = h$ and $o(v, p) \in [n - f + 1, n]$ (conversely if h was a female founder).

Instead of working with 2^{n-f} equivalence classes over the elements in $\mathbb{Z}_2^n$ we associate each equivalence class with an element of $\mathbb{Z}_2^{n-f}$. We call elements of $\mathbb{Z}_2^{n-f}$ *founder reduced* inheritance vectors. Using the inheritance vector layout, o, defined above we let any founder reduced inheritance vector $x = (x_1, \ldots, x_{n-f})$ represent the equivalence class $[\![(x_1, \ldots, x_{n-f}, 0, \ldots, 0)]\!]_S$ (x padded with f zeros). In this case we say that x represents its *natural* equivalence class. For the rest of this section all founder reduced inheritance vectors represent their natural equivalence classes. Furthermore, the founder reduced probability vector, $\widetilde{P}$, is given such that:

$$\widetilde{P}((x_1, \ldots, x_{n-f})) = 2^f \cdot P((x_1, \ldots, x_{n-f}, \overbrace{0, \ldots, 0}^{f})). \tag{4-6}$$

Example 25 (Representing Inheritance Vectors Symbolically) *For the pedigree of Figure 4-1 on page 41, defining an inheritance vector layout,* o*, on* (x_1, x_2, x_3, x_4)*, such that the last two bits represent one meiosis for both* $father$ *and* $mother$*, can be achieved letting* x_1 *correspond to* (son, p) *(*$o(son, p) = 1$*),* x_2 *to* (son, m) *(*$o(son, m) = 2$*),* x_3 *to* $(daughter, p)$ *(*$o(daughter, p) = 3$*), and* x_4 *to* $(daughter, m)$ *(*$o(daughter, m) = 4$*). Complete inheritance vectors are given as elements of* $\mathbb{Z}_2^4$*, whereas founder reduced inheritance vectors are elements of* $\mathbb{Z}_2^2$*. According to (4-6) the four equivalence classes represented by the founder reduced inheritance vectors are:*

$$\begin{aligned}
(0,0) \quad &\textit{represents} \quad [\![(0,0,0,0)]\!]_S = \{(0,0,0,0), (1,0,1,0), (0,1,0,1), (1,1,1,1)\}, \\
(0,1) \quad &\textit{represents} \quad [\![(0,1,0,0)]\!]_S = \{(0,1,0,0), (1,1,1,0), (0,0,0,1), (1,0,1,1)\}, \\
(1,0) \quad &\textit{represents} \quad [\![(1,0,0,0)]\!]_S = \{(1,0,0,0), (0,0,1,0), (1,1,0,1), (0,1,1,1)\}, \ \textit{and} \\
(1,1) \quad &\textit{represents} \quad [\![(1,1,0,0)]\!]_S = \{(1,1,0,0), (0,1,1,0), (1,0,0,1), (0,0,1,1)\}.
\end{aligned} \tag{4-7}$$

All founder reduced inheritance vectors represent their natural equivalence classes. Note that none of the elements on the right hand side are in more than one equivalence class, since one bit for each founder is fixed (the last two bits) for the elements which define the equivalence classes. The elements in the equivalence classes are listed in order such that the first element shown is the element itself, the second is the element added with the bit mask of $father$*,*

$(1,0,1,0)$, *the third is the element added with the bit mask of* $mother$, $(0,1,0,1)$, *and the fourth is the element added with both the bit masks of* $father$ *and* $mother$, $(1,1,1,1)$.

Example 24 motivated the correctness of Equation 4-5. For the current example, with the complete inheritance vector ordered such that the last bits represent one meiosis for each founder, the pairs with equal transition probabilities between equivalence classes are easily identified in (4-7). Take for instance the equivalence classes $[\![(0,0,0,0)]\!]_S$ *and* $[\![(0,1,0,0)]\!]_S$, *the pairs are then listed under each other in the sense that the transition probability between* $(1,0,1,0)$ *and* $(1,1,1,0)$ *is identical to that between* $(0,1,0,1)$ *and* $(0,0,0,1)$, *that between* $(1,1,1,1)$ *and* $(1,0,1,1)$, *and finally that between* $(0,0,0,0)$ *and* $(0,1,0,0)$. *In general if we take some transition probability* $T((1,1,1,0),(1,0,0,1))$ *(from equivalence classes* $[\![(0,1,0,0)]\!]_S$ *and* $[\![(1,1,0,0)]\!]_S$, *respectively) and we want to find the identical transition probability from* $(1,0,1,1)$ *to some element in* $[\![(1,1,0,0)]\!]_S$, *we notice that adding* $(1,1,1,0)$ *by the bit mask of* $mother$ *we get* $(1,0,1,1)$, *thus the element in* $[\![(1,1,0,0)]\!]_S$ *which we are searching for is simply* $(1,0,0,1)$ *added with the bit mask of* $mother$, *that is* $T((1,1,1,0),(1,0,0,1)) = T((1,0,1,1),(1,1,0,0))$.

In the literature founder reduction is sometimes referred to as *fixing bits* of the inheritance vector, [Gud00]. If a founder reduced inheritance vector, $x = (x_1,\ldots,x_{n-f})$, represents the natural equivalence class:

$$[\![(x_1,\ldots,x_{n-f},\overbrace{0,\ldots,0}^{f})]\!]_S,$$

we then say that $(x_1,\ldots,x_{n-f},0,\ldots,0)$ is the *representative element* of the equivalence class represented by x, and we denote it by v_x. Note that $x \in \mathbb{Z}_2^{n-f}$ and $v_x \in \mathbb{Z}_2^n$.

4.2.1 Recombination Transform on Founder Reduced Inheritance Vectors

Applying founder reduction to inheritance vectors introduces problems in multi point probability computation, since we need to consider all inheritance vectors in an equivalence class when transforming the founder reduced probability vector. We refer to [KL98, Gud00] for an explanation of how to perform Fourier transforms on founder reduced probability vectors and the corresponding transition matrix.

The reason that the recombination transform can be stated recursively is that we can compare the corresponding bits of two inheritance vectors v and w independently to determine the value of $T(v,w)$[8]. In other words, the fact that a recombination has occurred at meiosis i is independent of whether a recombination has occurred at any other meiosis. However, this is not the case for founder reduced inheritance vectors, since each of these represents a whole equivalence class.

The efficiency of the recombination transform lies in the decomposition (See Theorem 5). Performing founder reduction on probability vectors ruins part of the recursive structure of the transform. That is, it cannot be given as a Kronecker product of 2×2-matrices. To change the transform, such that it regains the recursive structure, we need a more thorough understanding

[8] For more on the transition matrix T see (3-4) on page 26.

of what it means that a transform can be given as a Kronecker product of 2×2-matrices. In the following we investigate the recursive structure of Kronecker products, before presenting the recombination transform of founder reduced probability vectors.

Kronecker Products

For matrices of dimension $2^n \times 2^n$ we define quadrants, such the first quadrant is the sub-matrix spanning the rows and columns 0 to $2^{n-1} - 1$. The second quadrant spans rows 0 to $2^{n-1} - 1$ and columns 2^{n-1} to $2^n - 1$. The third quadrant spans rows 2^{n-1} to $2^n - 1$ and columns 0 to $2^{n-1} - 1$. The fourth quadrant spans rows and columns 2^{n-1} to $2^n - 1$. This scheme is depicted in Figure 4-3.

$$\left[\begin{array}{c|c} 1 & 2 \\ \hline 3 & 4 \end{array}\right]$$

Figure 4-3: Illustrating the four quadrants of a matrix.

Observing that the i-th element of the Kronecker product is the matrix $\left[\begin{smallmatrix} (1-\theta) & \theta \\ \theta & (1-\theta) \end{smallmatrix}\right]$ means that for any two inheritance vectors v and w, the transition matrix for complete inheritance vectors $T(v, w)$ has $(1 - \theta)$ as a factor if the i-th bits of v and w are identical, and θ if they are not. The result of changing the i-th matrix to be $\left[\begin{smallmatrix} \theta & (1-\theta) \\ (1-\theta) & \theta \end{smallmatrix}\right]$ would be that $T(v, w)$ would have θ as a factor if the i-th bits of v and w were identical, and $(1 - \theta)$ if not.

From the definition of the Kronecker product we easily see why this is the case. If the first 2×2-matrix is the standard recombination transform then all elements of the first and fourth quadrants are multiplied by $(1 - \theta)$. These two quadrants are precisely the quadrants where the entry for two inheritance vectors (v, w) have equal values for their first bit. In quadrant one, the first bits of both v and w are 0 and in quadrant four the first bits of both v and w are 1. For the second 2×2-matrix of the Kronecker product we use the same reasoning this time on sub-quadrants of each of the four quadrants etc. Obviously, this only holds when the natural order of probability vectors and the transition matrices is assumed.

Example 26 (Kronecker Elements) *Consider the following two Kronecker products for inheritance vectors over $\mathbb{Z}_2^2$:*

$$A = \begin{bmatrix} (1-\theta) & \theta \\ \theta & (1-\theta) \end{bmatrix} \otimes \begin{bmatrix} (1-\theta) & \theta \\ \theta & (1-\theta) \end{bmatrix}$$
$$B = \begin{bmatrix} (1-\theta) & \theta \\ \theta & (1-\theta) \end{bmatrix} \otimes \begin{bmatrix} \theta & (1-\theta) \\ (1-\theta) & \theta \end{bmatrix}$$

A and B have the following form:

$$\begin{array}{cc} A & B \\ \begin{bmatrix} (1-\theta)^2 & (1-\theta)\theta & \theta(1-\theta) & \theta^2 \\ (1-\theta)\theta & (1-\theta)^2 & \theta^2 & \theta(1-\theta) \\ \theta(1-\theta) & \theta^2 & (1-\theta)^2 & (1-\theta)\theta \\ \theta^2 & \theta(1-\theta) & (1-\theta)\theta & (1-\theta)^2 \end{bmatrix} & \begin{bmatrix} (1-\theta)\theta & (1-\theta)^2 & \theta^2 & \theta(1-\theta) \\ (1-\theta)^2 & (1-\theta)\theta & \theta(1-\theta) & \theta^2 \\ \theta^2 & \theta(1-\theta) & (1-\theta)\theta & (1-\theta)^2 \\ \theta(1-\theta) & \theta^2 & (1-\theta)^2 & (1-\theta)\theta \end{bmatrix} \end{array}$$

For both A and B any entry, $((x_1, x_2), (y_1, y_2))$ (A and B are transition matrices over $\mathbb{Z}_2^n$) has $(1-\theta)$ as a factor when $x_1 = y_1$ and θ when $x_1 \neq y_1$. If for the second bit $x_2 = y_2$, then A has $(1-\theta)$ as factor, whereas B has θ and vice versa if the $x_1 \neq y_1$.

In the remaining part of this section we make use of the following facts about the Kronecker product, where A, B, B', and C are matrices for which B and B' have equal dimensions, and c is a scalar:

$$\begin{aligned} A \otimes (B + B') &= A \otimes B + A \otimes B' && \text{(4-8)} \\ A \otimes (B \otimes C) &= (A \otimes B) \otimes C && \text{(4-9)} \\ A \otimes cB = cA \otimes B &= c(A \otimes B). && \text{(4-10)} \end{aligned}$$

'+' denotes componentwise sum of matrices of equal dimension.

The Founder Reduced Recombination Transform

Equation (4-5) on page 44 states that: For any inheritance vector, v, in an equivalence class, $[\![v_x]\!]_S$, under founder reduction, if there is a transition probability $T(v, w) = p$ to some element, w, in another equivalence class, $[\![w_y]\!]_S$, then every other element in $[\![v_x]\!]_S$ also has transition probability p to some element in $[\![w_y]\!]_S$. This means that we need only consider the transform of all inheritance vectors in a equivalence class to the representative element in each equivalence class, as opposed to considering the transition probabilities between all inheritance vectors in each equivalence classes. Since there are 2^f inheritance vectors in each equivalence class, we can define each of the entries in the transition matrix, $\widetilde{T}$, between founder reduced inheritance vectors (and thus equivalence classes) as a sum of 2^f transition probabilities. That is, for any entry, $\widetilde{T}((x_1, \ldots, x_{n-f}), (y_1, \ldots, y_{n-f}))$, in the transition matrix between founder reduced inheritance vectors, should represent all 2^f transition matrix entries $T((x_1, \ldots, x_{n-f}, 0, \ldots, 0), y)$ where $y \in [\![(y_1, \ldots, y_{n-f}, 0, \ldots, 0)]\!]_S$ and T is the transition matrix over complete inheritance vectors. We first show how to compute the correct values for the entries in $\widetilde{T}$, we then show how to rewrite this computation in order to obtain an efficient algorithm.

We know that all inheritance vectors in an equivalence class $[\![v_y = (y_1, \ldots, y_{n-f}, 0, \ldots, 0)]\!]_S$ represented by the founder reduced inheritance vector $y = (y_1, \ldots, y_{n-f})$, can be generated from v_y by adding v_y with some element in the set, S, generated by the set of founder bit masks. We can then view the transition matrix $\widetilde{T}$ as the componentwise sum of 2^f transition matrices, one for each $t \in S$. Any entry (x, y) in the transition matrix for some $t \in S$ then holds the transition probability $T(v_x, v_y + t)$[9].

[9] Recall that $v_x = (x_1, \ldots, x_{n-f}, 0, \ldots, 0)$ and $v_y = (y_1, \ldots, y_{n-f}, 0, \ldots, 0)$ are the representative elements of the founder reduced inheritance vectors $x = (x_1, \ldots, x_{n-f})$ and $y = (y_1, \ldots, y_{n-f})$, respectively.

For each of the 2^f ways of generating the elements of an equivalence class from the representative element we define an tuple $\langle q, I\rangle$, where $q \in \mathbb{N}$ is the number of founders for which the bits concerning meiosis of those founders are switched, and I is the set of indices, i, in the interval $1 \leq i \leq n$, that concern meiosis for the q founders. Obviously, there is a one-to-one mapping from $\langle q, I\rangle$-tuples to elements of S. For each $t \in S$ the corresponding $\langle q, I\rangle$-tuple is given such that q refers to the number of founder bit masks which generated t and I is the set of indices for which t has bit value 1. We choose to define this information in $\langle q, I\rangle$-tuples since it later simplifies the founder reduced recombination transform.

Since there a mapping between the elements of S and $\langle q, I\rangle$-tuples, we know that any element in an equivalence class $[\![v_y]\!]_S$ can be generated from some $\langle q, I\rangle$-tuple. If an element w is generated from some representative element v_y using some $\langle q, I\rangle$-tuple, we say the w is the *mutation* of v_y with respect to $\langle q, I\rangle$. For this reason we refer to $\langle q, I\rangle$-tuples as *mutation tuples*.

Definition 15 (Mutation Tuples) *Given a pedigree with f founders and complete inheritance vectors over $\mathbb{Z}_2^n$. Let $BM = \{s_1, \ldots, s_f\}$ be the set of founder bit masks, where for any $X \subseteq BM$, $index(X)$ determines a set of indices, i, such that $i \in index(X)$ iff any $s_j \in X$ has bit value 1 on index i. Then the set of mutation tuples is given as:*

$$\mathcal{I} = \{\ \langle q, I\rangle \mid 0 \leq q \leq f \text{ and } I = index(X) \text{ where } X \subseteq BM \text{ with } |X| = q\}.$$

Obviously, $\mathcal{I}$ is the set of all 2^f $\langle q, I\rangle$-tuples described above, which generate an equivalence class from the representative element. Actually, any element in an equivalence class is the mutation of any other element with respect to some mutation tuple, $\langle q, I\rangle \in \mathcal{I}$.

Example 27 (Mutation Tuples) *Example 25 on page 46 shows the founder reduced inheritance vectors for the pedigree in Figure 4-1 on page 41 as:*

$(0,0)$ *represents* $[\![(0,0,0,0)]\!]_S = \{(0,0,0,0),(1,0,1,0),(0,1,0,1),(1,1,1,1)\}$,
$(0,1)$ *represents* $[\![(0,1,0,0)]\!]_S = \{(0,1,0,0),(1,1,1,0),(0,0,0,1),(1,0,1,1)\}$,
$(1,0)$ *represents* $[\![(1,0,0,0)]\!]_S = \{(1,0,0,0),(0,0,1,0),(1,1,0,1),(0,1,1,1)\}$, *and*
$(1,1)$ *represents* $[\![(1,1,0,0)]\!]_S = \{(1,1,0,0),(0,1,1,0),(1,0,0,1),(0,0,1,1)\}$.

The inheritance vector layout of the bits in the inheritance vector are as in Example 25 $(x_1, x_2, x_3, x_4) = ((son, p), (son, m), (daughter, p), (daughter, m))$. For the complete inheritance vectors, the four different mutation tuples are $\langle 0, \emptyset\rangle$, $\langle 1, \{1,3\}\rangle$, $\langle 1, \{2,4\}\rangle$, and $\langle 2, \{1,3,2,4\}\rangle$, counting indices from 1 to 4. This means that the first element shown in the equivalence class is the representative element for which no bits have been switched (the element itself). The second is the representative element with bits switched for one of the last two bits and index 1 (adding the bit mask of $father$). The third is the representative element with bits switched for one of the last two bits and index 2 (adding the bit mask of $mother$). The fourth is the representative element with bits switched for both of the founders (adding the bit masks of both $father$ and $mother$).

As stated previously, the entry $\widetilde{T}(x, y)$ in the transition matrix over founder reduced inheritance vectors should hold the sum $\sum_u T(v_x, u)$, where $u \in [\![v_y]\!]_S$ and T is the transition matrix between complete inheritance vectors. We know that $\widetilde{T}$ can be split up into the componentwise sum 2^f matrices, one for each $t \in S$. Since there is a mapping between the elements of S and the set of mutation tuples, we can similarly split up the transition matrix with respect to mutation tuples. We show this in the following example.

Example 28 (Splitting Up the Founder Reduced Transformation Matrix) *We show how to split up the transformation matrix for the founder reduced inheritance vectors for the ongoing example over the pedigree in Figure 4-1 on page 41. Example 27 on the facing page shows the founder reduced inheritance vectors and the equivalence classes which they represent. The transformation matrix for the founder reduced inheritance vectors can be split up into to the componentwise sum of four transformation matrices, one for each $\langle q, I\rangle$-tuple:*

$\langle 0, \emptyset\rangle :$	$(1-\theta)^2 \cdot \begin{bmatrix} (1-\theta)^2 & (1-\theta)\theta & \theta(1-\theta) & \theta^2 \\ (1-\theta)\theta & (1-\theta)^2 & \theta^2 & \theta(1-\theta) \\ \theta(1-\theta) & \theta^2 & (1-\theta)^2 & (1-\theta)\theta \\ \theta^2 & \theta(1-\theta) & (1-\theta)\theta & (1-\theta)^2 \end{bmatrix}$
$\langle 1, \{1,3\}\rangle :$	$\theta(1-\theta) \cdot \begin{bmatrix} \theta(1-\theta) & \theta^2 & (1-\theta)^2 & (1-\theta)\theta \\ \theta^2 & \theta(1-\theta) & (1-\theta)\theta & (1-\theta)^2 \\ (1-\theta)^2 & (1-\theta)\theta & \theta(1-\theta) & \theta^2 \\ (1-\theta)\theta & (1-\theta)^2 & \theta^2 & \theta(1-\theta) \end{bmatrix}$
$\langle 1, \{2,4\}\rangle :$	$(1-\theta)\theta \cdot \begin{bmatrix} (1-\theta)\theta & (1-\theta)^2 & \theta^2 & \theta(1-\theta) \\ (1-\theta)^2 & (1-\theta)\theta & \theta(1-\theta) & \theta^2 \\ \theta^2 & \theta(1-\theta) & (1-\theta)\theta & (1-\theta)^2 \\ \theta(1-\theta) & \theta^2 & (1-\theta)^2 & (1-\theta)\theta \end{bmatrix}$
$\langle 2, \{1,3,2,4\}\rangle :$	$\theta^2 \cdot \begin{bmatrix} \theta^2 & \theta(1-\theta) & (1-\theta)\theta & (1-\theta)^2 \\ \theta(1-\theta) & \theta^2 & (1-\theta)^2 & (1-\theta)\theta \\ (1-\theta)\theta & (1-\theta)^2 & \theta^2 & \theta(1-\theta) \\ (1-\theta)^2 & (1-\theta)\theta & \theta(1-\theta) & \theta^2 \end{bmatrix}$

For every transformation matrix there is a factor which is the same for every entry. In the above table this factor has been moved out of the matrix. We verify that the transition probabilities from the representative element, v_x, of $[\![(0,0,0,0)]\!]_S$ to each of the elements of $[\![1,0,0,0]\!]_S$ are given in entry $((0,0),(1,0))$ of the $\langle q, I\rangle$-transformation matrices. We show only one of the transition probabilities and urge the reader verify the rest. The transition probability $T((0,0,0,0),(1,1,0,1))$ is $\theta(1-\theta)^3$, and since $(1,1,0,1)$ is the mutation of the representative element, $(1,0,0,0)$, with respect to $\langle 1, \{1,3\}\rangle$, we need to very that the transition probability *in entry $((0,0),(1,0))$ of the $\langle 1,\{1,3\}\rangle$-transformation matrix is, indeed, $\theta(1-\theta)^3$. This entry has $(1-\theta)^2$ which multiplied by the constant $\theta(1-\theta)$ gives $\theta(1-\theta)^3$.*

As staed in Example 28, when the transformation matrix over founder reduced inheritance vectors are split up into 2^f transformation matrices, each of these can be given as a constant multiplied by a transformation matrix. The exact value of the constant in a $\langle q,I\rangle$-transformation matrix can be deduced from q. Recall that q represents the number of founder bit masks which have been added to the representative element to generate a member of the equivalence class. Since the last f bits of each representative element has bit value 0 and represent one meiosis for each founder, the newly generated element will have bit value 1 for q of the last f bits. This is the case because each of the q founder bit masks has bit value 1 for one of last f bits. This implies that the transition probability between the representative element v_x and the mutation, w, of some representative element v_y with respect to $\langle q,I\rangle$ has $\theta^q(1-\theta)^{f-q}$ as a factor because v_x has bit value 0 for the last f bits, whereas w has bit value 1 for q of the last f bits.

Example 29 (q Determines the Constant) *We easily verify that the constant for each of $\langle q,I\rangle$-transformation matrices in Examples 28 is given as $\theta^q(1-\theta)^{f-q}$. $\langle 0,\emptyset\rangle$ has constant $\theta^0(1-\theta)^2$, $\langle 1,\{1,3\}\rangle$ has constant $\theta(1-\theta)$, $\langle 1,\{2,4\}\rangle$ has constant $\theta(1-\theta)$, and $\langle 2,\{1,3,2,4\}\rangle$ has constant $\theta^2(1-\theta)^0$.*

Actually, with the 2^f $\langle q,I\rangle$-transformation matrices we have gained a recursive structure of the recombination transform on founder reduced inheritance vectors. Each of the $\langle q,I\rangle$-matrices without the constant can be rewritten as a Kronecker product of $n-f$ 2×2-matrices. The 2×2-matrices used are the normal recombination transform[10] and the *reverse* recombination transform of dimension 2×2:

$$\begin{bmatrix} \theta & (1-\theta) \\ (1-\theta) & \theta \end{bmatrix}. \tag{4-11}$$

Analogous to the recombination transform, we denote by $\overline{\mathcal{R}}_i^\theta$ the Kronecker product of i 2×2 reverse recombination transforms.

Definition 16 (Reverse Recombination Transform) *The reverse recombination transform of order n over recombination fraction θ is defined as:*

$$\overline{\mathcal{R}}_1^\theta = \begin{bmatrix} \theta & (1-\theta) \\ (1-\theta) & \theta \end{bmatrix};\ \overline{\mathcal{R}}_n^\theta = \begin{bmatrix} \theta\cdot\overline{\mathcal{R}}_{n-1}^\theta & (1-\theta)\cdot\overline{\mathcal{R}}_{n-1}^\theta \\ (1-\theta)\cdot\overline{\mathcal{R}}_{n-1}^\theta & \theta\cdot\overline{\mathcal{R}}_{n-1}^\theta \end{bmatrix}$$

Recall that the entry $(x=(x_1,\dots,x_{n-f}), y=(y_1,\dots,y_{n-f}))$ in any $\langle q,I\rangle$-matrix holds the transition probability $T(v_x,w)$ where w is the mutation of v_y with respect to $\langle q,I\rangle$. Above

[10] See Definition 12 on page 39 for the definition of the recombination transform.

we saw that the factor of the transition probability with respect the last f bits is given by q as $\theta^q(1-\theta)^{f-q}$. The factor of the transition probability with respect to the first $n-f$ bits depends on a bitwise comparison of the first $n-f$ bits between v_x and w. Recall that the i-th 2×2-matrix of a Kronecker product for vectors determines a common factor of all entries for which the i-th bits either are equal or different. E.g. if the i-th matrix is the reverse recombination transform then for the entry of any two vectors for which the i-th bits are either both 0 or 1, θ is a factor, and $(1-\theta)$ is a factor if the i-th bits are unequal. Since the mutation of v_y with respect to $\langle q, I\rangle$ is represented by $y = (y_1, \ldots, y_{n-f})$ in the $\langle q, I\rangle$-matrix, we cannot just perform a bitwise comparison between x and y.

However, the answer lies within I. If index i belongs to I then the i-th bit of w has the opposite value of the i-th bit of y, and if i does not belong to I then the i-th bit of w has the same value as the i-th bit of y. This follows from the definition of the elements in I, since an index i is in I if the i-th bit belongs to a bit mask of one of the founders which gave rise to $\langle q, I\rangle$.

Using this knowledge, we can define any $\langle q, I\rangle$-matrix as the Kronecker product $n-f$ 2×2-matrices. From any entry (x, y) we can deduce the factor for the first $n-f$ bits of the probability transform between v_x and w, where w is the mutation of v_y with respect to $\langle q, I\rangle$. For the i-th bit there are four different cases to consider:

1 $x_i = y_i$ and $i \in I$,

2 $x_i = y_i$ and $i \notin I$,

3 $x_i \neq y_i$ and $i \in I$,

4 $x_i \neq y_i$ and $i \notin I$.

From 1) we deduce that the i-th bits of v_x and w are unequal since i belongs to I which implies that the i-th bit of v_y has been switched in w. From 2) we deduce that the i-th bits of v_x and w are equal, since the i-th bit of v_y has not been switched in w. From 3) we deduce that the i-th bits of v_x and w are equal. Finally, from 4) we deduce that the i-th bits of v_x and w are unequal.

Since each $\langle q, I\rangle$-matrix can be stated as the Kronecker product of 2×2-matrices the decomposition theorem (Theorem 5 on page 40) applies, and we can compute each of the 2^f transforms recursively.

It should now be clear that any $\langle q, I\rangle$-matrix is identical to a matrix defined in terms of a $\langle q, I\rangle$-tuple as the Kronecker product of $n-f$ 2×2-matrices, such that the i-th element of the Kronecker product is the recombination transform if i does not belong to I and the reverse recombination transform if it does. For this purpose we define the *hybrid recombination transform*:

Definition 17 (Hybrid Recombination Transform) *Let $\langle q, I\rangle$ be a tuple, such that $0 \leq q \leq f$ and I is a set of indices, i, with $1 \leq i \leq n$, and θ a recombination fraction, then the hybrid* *recombination transform,* $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}$ *of order* $n-f$ *over* $\langle q,I\rangle$ *and* θ *is defined as:*

$$\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle} = \theta^{q}(1-\theta)^{f-q}\cdot\bigotimes_{j=1}^{n-f} R_j,$$

where

$$R_j = \begin{cases} \begin{bmatrix} \theta & (1-\theta) \\ (1-\theta) & \theta \end{bmatrix} & : j \in I \\ \begin{bmatrix} (1-\theta) & \theta \\ \theta & (1-\theta) \end{bmatrix} & : j \notin I \end{cases}$$

Obviously, any $\langle q,I\rangle$-matrix is identical to the hybrid recombination transform of order $n-f$ over $\langle q,I\rangle$ and the recombination fraction θ, that is $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}$. Furthermore, the recombination transform of founder reduced inheritance vectors is then given as the componentwise sum of 2^f hybrid recombination transforms, one form each $\langle q,I\rangle$.

Example 30 ($\langle q,I\rangle$-Matrices as Kronecker Products) *Each of the four* $\langle q,I\rangle$*-matrices of Example 28 can be given as the following Kronecker products of two* 2×2*-matrices:*

$\langle 0,\emptyset\rangle$:	$(1-\theta)^2\cdot\begin{bmatrix}(1-\theta) & \theta \\ \theta & (1-\theta)\end{bmatrix}\otimes\begin{bmatrix}(1-\theta) & \theta \\ \theta & (1-\theta)\end{bmatrix}$
$\langle 1,\{1,3\}\rangle$:	$\theta(1-\theta)\cdot\begin{bmatrix}\theta & (1-\theta) \\ (1-\theta) & \theta\end{bmatrix}\otimes\begin{bmatrix}(1-\theta) & \theta \\ \theta & (1-\theta)\end{bmatrix}$
$\langle 1,\{2,4\}\rangle$:	$(1-\theta)\theta\cdot\begin{bmatrix}(1-\theta) & \theta \\ \theta & (1-\theta)\end{bmatrix}\otimes\begin{bmatrix}\theta & (1-\theta) \\ (1-\theta) & \theta\end{bmatrix}$
$\langle 2,\{1,3,2,4\}\rangle$:	$\theta^2\cdot\begin{bmatrix}\theta & (1-\theta) \\ (1-\theta) & \theta\end{bmatrix}\otimes\begin{bmatrix}\theta & (1-\theta) \\ (1-\theta) & \theta\end{bmatrix}$

The Kronecker product for the $\langle 0,\emptyset\rangle$*-matrix is given as two normal recombination transforms since it represents the mutation of representative elements with respect to no founder bit masks, that is the element itself. The* $\langle 1,\{1,3\}\rangle$*-matrix represents elements for which the first and third bits of the representative element has been switched, which means that if for any entry* (x,y)*,* x *and* y *have equal values for their first bits, then the mutation of* v_y *with respect to* $\langle 1,\{1,3\}\rangle$ *and* v_x *have unequal values for their first bits. Thus the first matrix of the Kronecker product is the reverse recombination transform and the second the normal recombination transform. For* $\langle 1,\{2,4\}\rangle$ *the second and fourth bits are being switched and the second* 2×2*-matrix is the reverse recombination transform. For* $\langle 2,\{1,3,2,4\}\rangle$ *all bits are being switched and both matrices of the Kronecker product are the reverse recombination transform.*

We are now able to give the theorem stating how we can perform the recombination transform over founder reduced inheritance vectors.

Theorem 7 (Recombination Transform of Founder Reduced Inheritance Vectors) *Consider a pedigree with $|N|$ non-founders and f founders with recombination fraction θ between two markers and inheritance vectors over $\mathbb{Z}_2^n$ ($n = 2 \cdot |N|$). Let P be the probability vector over complete inheritance vectors and $\widetilde{P}$ the probability vector over founder reduced inheritance vectors. Now let C be the recombination transform of P, ($C = \mathcal{R}_n^\theta P$), and $\mathcal{I}$ be the set of mutation tuples, then $\widetilde{C}$ is the founder reduced recombination transform of $\widetilde{P}$:*

$$\widetilde{C} = \sum_{\langle q,I\rangle \in \mathcal{I}} \mathsf{R}_{n-f}^{\theta,\langle q,I\rangle} \cdot \widetilde{P},$$

where the summation is componentwise summation of matrices. Then, for C and $\widetilde{C}$ the following holds:

$$\forall v \in [\![v_x]\!]_S \,.\, C(v) = \frac{1}{2^f} \cdot \widetilde{C}(x). \tag{4-12}$$

The arguments for the proof of Theorem 7 has basically been stated through the examples in this section, but we include the formal proof for completion.

Proof: To prove that $C(v) = \frac{1}{2^f}\widetilde{C}(x)$ it suffices to show that it holds for the representative element, v_x, of $x = (x_1, \ldots, x_{n-f})$. This suffices since Equation 4-5 on page 44 states that any two inheritance vectors in an equivalence class have equal probabilities after the recombination transform.

The proof has two parts; first, we prove that for any element that makes up the transform, $C(v_x)$, there is a unique identical element in the transform $\widetilde{C}(x)$. Second, we prove that for every element that makes up the transform, $\widetilde{C}(x)$, there is an unique identical element in $C(v_x)$.
1):
From multi point probability computation Equation (3-5) on page 26 states that $C(v_x)$ is given as the contribution $\sum_w T(v_x, w)P(w)$ and we choose any such element, w, to prove that the value of $T(v_x, w)P(w)$ is an element in the founder reduced recombination transform, $\widetilde{C}(x)$. We know that w is a member of some equivalence class $[\![v_y = (y_1, \ldots, y_{n-f}, 0, \ldots, 0)]\!]_S$ and there exists a tuple $\langle q, I\rangle$ such that w is the mutation of v_y with respect to $\langle q, I\rangle$. Let $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}$ be the corresponding hybrid recombination transform. We now show that the entries $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}(x, y)$ and $T(v_x, w)$ are equal. We know that v_x and w are different for q of the last f bits and identical for $f - q$, which lead to a factor of $\theta^q(1-\theta)^{f-q}$ in $T(v_x, w)$ and this is exactly the constant of $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}$. For any index, i, of the first $n - f$ bits, if i is in I then v_x and w have different bit values if $x_i = y_i$ (and thus factor θ) and equal if $x_i \neq y_i$ (and factor $(1-\theta)$), which also holds for $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}$ since the i-th Kronecker matrix is the reverse recombination transform. Similarly, if $i \notin I$. Finally, since $P(w) = P(v_y)$ and $\widetilde{P}(y) = 2^f \cdot P(v_y)$ we have that $C(v_x) = \frac{1}{2^f}\widetilde{C}(x)$.
2):
Take any $\langle q, I\rangle \in \mathcal{I}$ and choose any equivalence class $[\![v_y = (y_1, \ldots, y_{n-f}, 0, \ldots, 0)]\!]_S$. We then need to show that entries $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}(x, y)$ and $T(v_x, w)$ are equal, where w is the mutation

of v_y with respect to $\langle q, I\rangle$. Proving this follows the same lines as the first part of the proof, and since we have that $\widetilde{P}(y) = 2^f \cdot P(v_y) = 2^f \cdot P(w)$, the second part is proven.
The theorem follows directly from 1) and 2). □

The complexity of performing the recombination transform on founder reduced inheritance vectors is $\mathcal{O}(2^f \cdot (n-f) \cdot 2^{n-f}) = \mathcal{O}((n-f)2^n)$. However, this is a worse complexity than that described by Kruglyak et. al in [KL98] where fast Fourier transform is applied to founder reduced inheritance vectors to achieve a complexity of $\mathcal{O}((n-f)2^{n-f})$. Similarly to inheritance vector which are not founder reduced, this complexity arises from transforming the founder reduced probability vector and the founder reduced transition matrix given as a vector, multiplying the resulting vectors, and then transforming the result back again. That is, three Fourier transforms and one multiplication.

Reducing Complexity of the Founder Reduced Recombination Transform

We now show how to reduced the complexity of the founder reduced recombination transform by rewriting the transform according to distributive law of the Kronecker product and componentwise sum of matrices, and the rule for constants. These are the rules stated as Equations (4-8) and (4-10).

In order to reduce the complexity of the founder reduced recombination transform, we first need to show why the bits concerning meiosis founders with a single child can be omitted in the transform.

Bits concerning founders with a single child:
We show how the componentwise sum of 2^f $\langle q, I\rangle$-matrices can be reduced to 2^{f-l} $\langle q, I\rangle$-matrices where l is number of founders which have only a single child. The result of a founder, h, having a single child is, that the bit representing the meiosis concerning founder h is positioned as one of the last f bits of the complete inheritance vector. Hence, for each $\langle q, I\rangle$-matrix there exists a matrix defined by $\langle q+1, I'\rangle$, where I' has the same elements as I with the addition of the index, k, for the meiosis concerning h. Since k is positioned as one of the last f bits, the hybrid recombination transforms $\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle}$ and $\mathsf{R}_{n-f}^{\theta,\langle q+1,I'\rangle}$ have different constants but identical matrices. We denote this matrix by X. By performing the componentwise sum of the two we arrive at:

$$\begin{aligned}
\mathsf{R}_{n-f}^{\theta,\langle q,I\rangle} + \mathsf{R}_{n-f}^{\theta,\langle q+1,I'\rangle} &= \theta^q(1-\theta)^{f-q}X + \theta^{q+1}(1-\theta)^{f-(q+1)}X \\
&= (\theta^q(1-\theta)^{f-q} + \theta^{q+1}(1-\theta)^{f-(q+1)})X \\
&= (\theta + (1-\theta))\theta^q(1-\theta)^{f-(q+1)}X \\
&= \theta^q(1-\theta)^{(f-1)-q}X.
\end{aligned}$$

It follows that we only need to consider *one* of the two transforms multiplied by the constant $\theta^q(1-\theta)^{(f-1)-q}$. Since for each founder who has only one child, we can consider half the number of matrices, then if there are l such founders we need to take the componentwise sum of just 2^{f-l} $\langle q, I\rangle$-matrices where we choose $\langle q, I\rangle$ such that q only refers to founders who have more than one child. Thus, we define the scalar of the $\langle q, I\rangle$-matrix to be $\theta^q(1-\theta)^{(f-l)-q}$.

For simplicity we now only consider the case where founders have more than one child so that we need not remember the constant, l.

Grouping bits into sequences:
By inducing further restrictions on the ordering of bits in the complete inheritance vectors, we will now explain how there is sharing among the 2^f transforms, such that we can achieve an upper bound complexity of $\mathcal{O}((n-f)2^{n-f})$, the same complexity as that of the fast Fourier transform-based method[11].

The required restriction is that bits in complete inheritance vectors need to be *grouped*, such that bits concerning the same founder meiosis are positioned in consecutive order, except for the one which is bound to one the last f bits. E.g. if a male founder has four children then the paternal bit of one of the children is position as one of the last f bits of the complete inheritance vector, and the paternal bits of the remaining three children are positioned in consecutive order in the vector, within the first $n-f$ bits.

Example 31 (Grouping of Bits Concerning Founder Meiosis) *In the preceding examples using the pedigree of Figure 4-1 on page 41, we assumed an ordering such that for a complete inheritance vector $(x_1, x_2, x_3, x_4) \in \mathbb{Z}_2^4$, x_1 represents (son, p), x_2 represents (son, m), x_3 represents $(daughter, p)$, and x_4 represents $(daughter, m)$. This ordering respects the grouping of bits just described since there are only two meiosis concerning each founder. Had we introduced a new child of $father$ and $mother$, called $sister$, then we would require an ordering like $(x_1, \ldots, x_6) \in \mathbb{Z}_2^6$, where x_1 represents (son, p), x_2 represents $(sister, p)$, x_3 represents (son, m), x_4 represents $(sister, m)$, x_5 represents $(daughter, p)$, and x_6 represents $(daughter, m)$.*

When inheritance vectors are grouped in this way, we can divide a founder reduced inheritance vector into sequences of consecutive bits such that a sequence contains either the bits not concerned with any founder meiosis, or *all* the bits concerning a founder meiosis. Assume that we have a founder reduced inheritance vector $x = (x_1, x_2, \ldots, x_{n-f})$, we can the divide the bits into sequences such that:

$$x = (\overbrace{x_1, \ldots, x_{k_1}}^{seq_1}, \overbrace{x_{k_1+1}, \ldots, x_{k_2}}^{seq_2}, \overbrace{x_{k_2+1}, \ldots, x_{k_3}}^{seq_3}, \ldots, \overbrace{x_{k_{(m-1)}+1}, \ldots, x_{k_m}}^{seq_m}), \tag{4-13}$$

where each sequence is either related to a single founder or no founder.

Rewriting the recombination transform:
Recall from Definition 7 on page 55 that the contribution over founder reduced inheritance vectors can be computed as the componentwise sum of hybrid recombination transforms:

$$\widetilde{C} = \sum_{\langle q,I\rangle \in \mathcal{I}} \mathsf{R}_{n-f}^{\theta,\langle q,I\rangle} \cdot \widetilde{P}. \tag{4-14}$$

[11] See Theorem 3 on page 35 for more on the complexity of the fast Fourier transform.

If we assume that the bits of the founder reduced inheritance vectors are grouped so that all meiosis concerning a founder are in consecutive order, we can rewrite (4-14) in order to obtain a more efficient algorithm. We do so by considering sequences of bits, where all the bits in a sequence either concern no founder meiosis, or where a sequence consists of all bits in the reduced inheritance vector concerning one founder. We denote the bits in a sequence by $x_1, \ldots, x_k$. We then need to consider two cases.

In the first case, where the sequence of k bits concern no founder, we observe that all the $\langle q, I\rangle$ matrices have same the 2×2 matrix for all of the k bits, since none of the indices $1, \ldots, k \notin I$ for any of the tuples. Hence, for the first k bits we have to perform the normal recombination transform, $\mathcal{R}_k^\theta$. If we use the distributive law for Kronecker products, thus that $A \otimes B + A \otimes C = A \otimes (B + C)$, where $+$ denotes the componentwise sum and where the matrices B and C have equal dimensions, we can rewrite (4-14):

$$\widetilde{C} = \sum_{\langle q,I\rangle \in \mathcal{I}} \mathsf{R}_{n-f}^{\theta,\langle q,I\rangle} \cdot \widetilde{P} \;=\; \mathcal{R}_k^\theta \otimes \sum_{\langle q',I'\rangle \in \mathcal{I}'} \mathsf{R}_{n-f-k}^{\theta,\langle q',I'\rangle} \cdot \widetilde{P}, \tag{4-15}$$

where $\mathcal{I}'$ denotes the set of $\langle q', I'\rangle$ tuples for which we have removed the first k bits, we have already accounted for those by moving the normal recombination transform out of the summation. $\mathcal{I}'$ contains 2^f tuples.

In the second case, where the sequence of bits are all the bits concerning some founder, we know that for half of the $\langle q, I\rangle$ matrices the indices $1, \ldots, k \in I$, while for the other half $1, \ldots, k \notin I$. Hence, we can rewrite (4-14):

$$\begin{aligned}
\widetilde{C} &= \sum_{\langle q,I\rangle \in \mathcal{I}} \mathsf{R}_{n-f}^{\theta,\langle q,I\rangle} \cdot \widetilde{P} \\
&= \Big((1-\theta)\mathcal{R}_k^\theta \otimes \sum_{\langle q'',I''\rangle \in \mathcal{I}''} \mathsf{R}_{n-f-k}^{\theta,\langle q'',I''\rangle} \cdot \widetilde{P}\Big) + \Big(\theta\bar{\mathcal{R}}_k^\theta \otimes \sum_{\langle q'',I''\rangle \in \mathcal{I}''} \mathsf{R}_{n-f-k}^{\theta,\langle q'',I''\rangle} \cdot \widetilde{P}\Big) \\
&= ((1-\theta)\mathcal{R}_k^\theta + \theta\bar{\mathcal{R}}_k^\theta) \otimes \sum_{\langle q'',I''\rangle \in \mathcal{I}''} \mathsf{R}_{n-f-k}^{\theta,\langle q'',I''\rangle} \cdot \widetilde{P},
\end{aligned} \tag{4-16}$$

where $\mathcal{I}''$ denotes the set of $\langle q'', I''\rangle$ tuples for which we have removed the first k bits *and* the founder which the first k bits concern, thus $\mathcal{I}''$ contains 2^{f-1} tuples. Since one founder has been removed the factor of the scalar concerning that founder, has also been moved out of the summation[12]. This is allowed due to the Kronecker rule (4-10) which states that $A \otimes cB = cA \otimes B = c(A \otimes B)$ if A and B are matrices and c is a scalar. Since every $\langle q, I\rangle$-matrix has an equivalent hybrid recombination transform[13], (4-10) allows us to move the scalar of the transform (or a factor of the scalar) onto any of the $n - f$ 2×2-matrices of the Kronecker product.

[12] We know that each founder gives rise to a factor of the scalar, namely $(1-\theta)$ if we have performed a founder switch for this founder and θ if not.

[13] Hybrid Recombination Transform: See Definition 17 on page 53.

Thus, we have $(1-\theta)$ as a factor for the $\langle q, I\rangle$ matrices where $1,\ldots,k \notin I$, which we know are half the total number of tuples. For the other half of the $\langle q, I\rangle$ matrices we have moved the factor θ out of the summation, since the bits concerning the founder have been switched. We know from Definition 17 on page 53 of the hybrid recombination transform that those $\langle q, I\rangle$ tuples for which I contains the bit mask for the founder, has the factor θ as part of the scalar, and similarly the tuples, for which I does not contain the bit mask for the founder, have $(1-\theta)$ as a factor of the scalar.

We can perform one of the two cases for all the sequences of bits in a founder reduced, grouped inheritance vector. When doing so we end up and with the Kronecker products of the normal recombination transform for the bits which concern no founders, and the sum of the normal and inverse recombination transforms multiplied by the appropriate scalars, for those groups of bits concerning a founder. The result of rewriting all groups is e.g. a formula of the form:

$$\widetilde{C} = \mathcal{R}^{\theta}_{k_1} \otimes ((1-\theta)\mathcal{R}^{\theta}_{k_2-k_1} + \theta\overline{\mathcal{R}}^{\theta}_{k_2-k_1}) \otimes ((1-\theta)\mathcal{R}^{\theta}_{k_3-k_2} + \theta\overline{\mathcal{R}}^{\theta}_{k_3-k_2}) \otimes \ldots \otimes \mathcal{R}^{\theta}_{k_m-k_{m-1}}, \tag{4-17}$$

if we assume that there are m sequences, and that none of the bits in the first sequence (bits at entries $1,\ldots,k_1$) concern any founder meiosis, while the second (entries $k_1+1,\ldots,k_2$), and the third (entries $k_2+1,\ldots,k_3$) sequence each concern a founder, etc.

Example 32 (Rewriting the Transform for Efficiency) *Continuing the ongoing example using the pedigree of Figure 4-1 on page 41, which has $\langle q, I\rangle$-matrices according to Example 30. We know that both bits in the founder reduced inheritance vector concern meiosis of the two founders, $father$ and $mother$. This means that we have two sequences of bits concerning a founder, each of length 1. Using the rewrite rules we obtain that the componentwise sum of the 4 $\langle q, I\rangle$-matrices can be given as:*

$$((1-\theta)\cdot\mathcal{R}^{\theta}_1 + \theta\cdot\overline{\mathcal{R}}^{\theta}_1) \otimes ((1-\theta)\cdot\mathcal{R}^{\theta}_1 + \theta\cdot\overline{\mathcal{R}}^{\theta}_1).$$

This approach to computing the contribution, $\widetilde{C}$, is more efficient than the componentwise sum of 2^f matrices multiplied by a probability vector as shown in Theorem 17 on page 53. The key is that for half the $\langle q, I\rangle$ tuples, the bits concerning a founder are switched, while for the other half they are not. By grouping bits we can simply handle one founder at a time, as shown above. We can now decompose with respect to the sum of the 2^f hybrid recombination transforms, when bits are grouped, to achieve a complexity of $\mathcal{O}((n-f)2^{n-f})$, where n is two times the number of non-founders in the pedigree, and f is the number of founders.

As with the recombination transform[14] we can state a decomposition theorem for the recombination transform over grouped, founder reduced inheritance vectors. The decomposition rule for the latter is, however, dependent on the *context* of the bit of the founder reduced inheritance vector, for which we are currently decomposing.

A bit can have one of three different contexts:

[14] See Definition 12 on page 39

1. A bit can represent a meiosis concerning no founder. Let Γ be the set of indices for bits which concern no founder.

2. A bit can be the *first* bit in a sequence concerning a founder. Let α be the set of indices for such bits.

3. A bit can be a bit in a sequence concerning a founder, but it is not the first bit in the sequence. Let β be the set of indices, except for the first, for bits in a sequence concerning some founder.

Hence, we have that if a founder reduced inheritance vector has a length n, then $\Gamma \cup \alpha \cup \beta = \{1, \ldots, n\}$, and that the intersection of any two of the sets is empty.

The decomposition rules for founder reduced, grouped inheritance vectors is stated below followed by an explanation of how to interpret it:

$$
\begin{aligned}
\Re_n^\theta \cdot \widetilde{P} &= \begin{cases}
\begin{bmatrix} (1-\theta) \cdot \Re_{n-1}^\theta \widetilde{P}_0 + \theta \cdot \Re_{n-1}^\theta \widetilde{P}_1 \\ \theta \cdot \Re_{n-1}^\theta \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^\theta \widetilde{P}_1 \end{bmatrix} : n \in \Gamma \\
(1-\theta) \cdot \begin{bmatrix} (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \\ \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \end{bmatrix} + \\
\theta \cdot \begin{bmatrix} \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \\ (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \end{bmatrix} : n \in \alpha
\end{cases} \\
\Re_n^{\theta,nor} \widetilde{P} &= \begin{cases}
\begin{bmatrix} (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \\ \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \end{bmatrix} : n \in \beta \\
\Re_n^\theta \cdot \widetilde{P} : \text{otherwise}
\end{cases} \\
\Re_n^{\theta,rev} \widetilde{P} &= \begin{cases}
\begin{bmatrix} \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \\ (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \end{bmatrix} : n \in \beta \\
\Re_n^\theta \cdot \widetilde{P} : \text{otherwise}
\end{cases}
\end{aligned}
$$

Each time a decomposition rule is applied, we decompose with respect to the positive and negative cofactors of the probability vector $\widetilde{P}$, hence we decompose with respect to a bit of the founder reduced inheritance vector. Recall that we require the bits of inheritance vectors to be grouped, thus we can always divide an inheritance vector into sequences of bits concerning no founder one sequence of bits for each founder.

Context 1: The bit represents a meiosis concerning no founder: ($n \in \Gamma$)
If a bit is in a sequence of bits not concerned with any founder meiosis, we decompose with respect the first decomposition rule for $\Re_n^\theta$:

$$
\Re_n^\theta \cdot \widetilde{P} = \begin{bmatrix} (1-\theta) \cdot \Re_{n-1}^\theta \widetilde{P}_0 + \theta \cdot \Re_{n-1}^\theta \widetilde{P}_1 \\ \theta \cdot \Re_{n-1}^\theta \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^\theta \widetilde{P}_1 \end{bmatrix},
$$

since all $\langle q, I \rangle$ matrices contain the normal recombination transform for such a bit, as shown in (4-15). When we have applied this decomposition rule for all of the bits in a sequence not concerning any founder meiosis, the result is equal to the decomposition of the normal recombination transform for l recursions:

$$\mathcal{R}_l^\theta,$$

where l is the length of the sequence.

Context 2: The bit represents the first bit concerning a founder ($n \in \alpha$):

If the bit is the first is a sequence concerning one founder, we apply the second decomposition rule for $\Re_n^\theta$:

$$\Re_n^\theta \cdot \widetilde{P} = (1-\theta) \cdot \overbrace{\begin{bmatrix} (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \\ \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \end{bmatrix}}^{Normal} + \theta \cdot \overbrace{\begin{bmatrix} \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \\ (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \end{bmatrix}}^{Reverse}$$

We perform both a normal recombination transform (marked $Normal$), and a reverse recombination transform ($Reverse$) for all of the bits in the sequence, *and* we multiply the results by their respective factors[15].

Context 3: The bit concerns a founder, but it is not the first bit in a sequence ($n \in \beta$):

When a bit is in a sequence concerning some founder, but not the first bit in this sequence, we apply two independent decompositions, one normal, $\Re^{\theta,nor}$, and one reversed, $\Re^{\theta,rev}$. The normal decomposition is given as follows:

$$\Re_n^{\theta,nor} \widetilde{P} = \begin{cases} \begin{bmatrix} (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \\ \theta \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,nor} \widetilde{P}_1 \end{bmatrix} : n \in \beta \\ \Re_n^\theta \cdot \widetilde{P} : \text{otherwise} \end{cases}$$

Note that we only perform this decomposition as long as we are in the *same* sequence, hence if the current bit is the first bit in a new sequence, concerning some founder or not, we use the decomposition rule for $\Re_n^\theta$.

Similarly for the reverse decomposition:

$$\Re_n^{\theta,rev} \widetilde{P} = \begin{cases} \begin{bmatrix} \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \\ (1-\theta) \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_0 + \theta \cdot \Re_{n-1}^{\theta,rev} \widetilde{P}_1 \end{bmatrix} : n \in \beta \\ \Re_n^\theta \cdot \widetilde{P} : \text{otherwise} \end{cases}$$

[15] See (4-16) page 58 for the explanation of these factors.

we only decompose with respect to this rule as long as we are in the current sequence. When the sequence ends both $\Re_i^{\theta,nor}$ and $\Re_i^{\theta,rev}$ revert back to $\Re_i^{\theta}$, and they share the common transform.

If we combine the decomposition rules for the bits in a sequence concerning a founder (context 2 and 3) the result of decomposing such a sequence is equal to:

$$((1-\theta)\mathcal{R}_l^{\theta} + \theta\overline{\mathcal{R}}_l^{\theta}),$$

where l is the length of the sequence.

When we have decomposed with respect to all the bits in all sequences we obtain a result similar to (4-17). Two decompositions are necessary for bits concerning a founder, and only one decomposition for bits concerning no founder meiosis.

The worst case complexity of the decomposition is the case where we have to perform two decompositions for each bit. Still, we have obtained a worst case complexity of $\mathcal{O}(2(n-f)2^{n-f}) = \mathcal{O}((n-f)2^{n-f})$, since whenever a sequence of founder-related bits ends, the normal and reverse decomposition revert back to the same transform and the result is shared. This way we are never performing more than two simultaneous transforms, hence the complexity.

4.3 Summary

In this chapter we proposed a new method for performing multi point probability computation. The method is based on the recombination transform. We showed how the recombination transform could be decomposed similarly to the Walsh-Hadamard transform to achieve an upper bound complexity of $\mathcal{O}(n \cdot 2^n)$, where n is two times the number of non-founders in the pedigree. The theoretical complexity of the recombination transform is identical to that of the Walsh-Hadamard transform, however, we showed how the recombination transform required just one transform and no multiplication to calculate contributions in multi point probability computation.

Furthermore, we showed how to modify the recombination transform to be used over founder reduced inheritance vector, thus achieving an upper bound complexity of $\mathcal{O}((n-f) \cdot 2^{n-f})$.

By applying the recombination transform over founder reduced inheritance vector we can perform multi point probability computation faster than using Fourier transforms.

Heuristic-based Representation

CHAPTER 5

In this chapter we evaluate whether linkage analysis can benefit from the use of DD-based data structures. Since the applicability of heuristic-based methods on a problem domain is dependent on their performance on real-life examples, our conclusions are drawn based on tests using real pedigrees. In the analysis we first go through our motivation behind the idea of using graph-based, symbolic representations in linkage analysis, then we lay out the strategy for our test. We then have a section on our test setup, a section on the different variable orderings used in the evaluation of the various DD-based data structures. Finally, we present the result of the tests.

However, for readers unfamiliar with DD-based data structures we, first, give the definition of *multi terminal binary decision diagrams* (MTBDDs). We choose MTBDDs, since the result of the following analysis shows that multi point probability computation can benefit from the compactness of graph-based data structures using the *Shannon decomposition rule*, the MTBDD is the simplest such data structure.

5.1 Multi-Terminal Binary Decision Diagrams

In this section we give a brief introduction to Multi-Terminal Binary Decision Diagrams (MTBDDs). We define the data structure and provide the semantics of nodes in terms of functions. We propose algorithms for performing single and multi point probability computations on MTBDD representations of probability distributions in Chapter 6 on page 89. These algorithms are defined in terms of the formal definition of MTBDDs provided in this section.

A definition and explanation of all the data structures used in this analysis is provided in Appendix B. Readers unfamiliar with DD-based data structures would benefit from reading the appendix, since it can provide some intuitive understanding of different graph-based data structures.

5.1.1 Introduction to MTBDDs

MTBDDs are a natural extension of *binary decision diagrams* (BDDs) as introduced by Bryant in 1986, [Bry86], which let terminals take non-binary values. BDDs are based on representing boolean functions using their representation as binary trees with the truth assignment as terminal values. Such a representation is exponential in the number of boolean variables. BDDs, as opposed to binary trees, are binary graphs for which common subgraphs are shared. This way BDDs provide canonical representations of all boolean functions.

Like BDDs, MTBDDs provide canonical representations for any function from $\mathbb{B}^n$ onto $\mathbb{R}$[1]. The idea of extending BDDs to allow non-binary terminals was introduced in 1993 by Fujita et al. in [FYC$^+$93] to perform Walsh transformations on BDDs. This article allowed terminals to represent the set of integers, whereas another article published the same year, [FYM93], allowed the terminals to represent reals, $\mathbb{R}$.

Formally, MTBDDs can be defined as follows:

Definition 18 (Multi Terminal Binary Decision Diagram) *A MTBDD over a set of variables VAR is a rooted, directed acyclic graph with node set $V \cup T$ and four functions:*

- $level : V \cup T \rightarrow \mathbb{N} \cup \{\infty\}$,
- $label : V \cup T \rightarrow$ *VAR* $\cup \mathbb{R}$,
- $child_0 : V \rightarrow V \cup T$,
- $child_1 : V \rightarrow V \cup T$, *such that:*

- *T is the set of terminal nodes n with $level(n) = \infty$ and $label(n) \in \mathbb{R}$.*
- *all other nodes $n \in V$ are inner nodes with $level(n) \in \mathbb{N}$, $label(n) \in$ VAR, and $child_0(n), child_1(n) \in V \cup T$.*
- *an MTBDD is ordered if every path from the root node to a terminal respect the level ordering of the variables, that is:*

$$\forall n, n' \in V \; . \; child_i(n) = n' \Rightarrow level(n) < level(n'), \tag{5-1}$$

  where $i \in \{0, 1\}$.

- *an MTBDD is reduced iff:*
  - *there are no redundant nodes, that is:*

$$\forall n \in V \; . \; child_0(n) \neq child_1(n), \tag{5-2}$$

  - *the graph has maximal sharing of subgraphs, that is:*

$$\begin{aligned} \forall n, n' \in V \quad . \quad & label(n) = label(n') \text{ and} \\ & level(n) = level(n') \text{ and} \\ & child_0(n) = child_0(n') \text{ and} \\ & child_1(n) = child_1(n') \Rightarrow n = n'. \end{aligned} \tag{5-3}$$

[1] BDD provide canonicity for the representation of functions from $\mathbb{B}^n$ onto $\mathbb{B}$.

To understand the semantics of MTBDDs in terms of functions we introduce the notion of *cofactors* and *Shannon decomposition*.

Given the function $g(x_1, \ldots, x_n)$ over n boolean variables onto $\mathbb{R}$, the *cofactor* $g_{x_i\overline{x_j}x_k}$ is the function obtained by setting $x_i = 1, x_j = 0, x_k = 1$ and evaluating g for $(1 \leq i, j, k \leq n,\ i \neq j \neq k)$[2].

Example 33 (Cofactors) *Consider a function g over three boolean variables $x_1, x_2,$ and x_3 defined as:*

x_1	x_2	x_3	$g(x_1, x_2, x_3)$
0	0	0	0
1	0	0	4
0	1	0	2
1	1	0	6
0	0	1	1
1	0	1	5
0	1	1	3
1	1	1	7

g corresponds to the function $g(x_1, x_2, x_3) = 4x_1 + 2x_2 + 1x_3$. The cofactor $g_{x_1\overline{x_3}}$ of g is the function:

x_2	$g_{x_1\overline{x_3}}(x_2)$
0	4
1	6

Using cofactors we can rewrite functions over boolean variables with the *Shannon decomposition* on any variable, x_i:

$$g = \overline{x_i} g_{\overline{x_i}} + x_i g_{x_i}, \tag{5-4}$$

where $g_{\overline{x_i}}$ and g_{x_i} are the *negative* and *positive* cofactors, respectively.

MTBDDs are simply the Shannon decomposition of functions with signature $g : \mathbb{B}^n \rightarrow \mathbb{R}$ and we define the function f of any node n[3] in a MTBDD as:

$$f(\langle n \rangle) = \begin{cases} label(n) : \ n \in T \\ (1 - x) \cdot f(\langle child_0 \rangle) + x \cdot f(\langle child_1(n) \rangle) : \ n \in V \text{ and } label(n) = x. \end{cases} \tag{5-5}$$

The canonicity of MTBDDs states that for any function $g : \mathbb{B}^n \rightarrow \mathbb{R}$ there exists exactly one MTBDD with root node n, such that $g = f(\langle n \rangle)$ for any assignment of the boolean variables.

[2] Notice that in Section 3.3.2 we introduced a similar notion for vectors.

[3] Not to be confused with the n in $\mathbb{B}^n$. We use n for both, since it should be obvious when it refers to a node and when it refers to the number of variables.

Example 34 (Function representation with MTBDDs) *The MTBDD representation of a function $g(x,y,z) = -2xyz + 10x + 2yz + 4y + 2z$ is given in Figure 5-1. According to Equation 5-5 the value of g(1,1,0) is given as:*

$$\begin{aligned} g(1,1,0) &= f(\langle n_1 \rangle) = 0 \cdot f(\langle child_0(n_1) \rangle) + 1 \cdot f(\langle child_1(n_1) \rangle) \\ &= 0 \cdot f(\langle child_0(n_3) \rangle) + 1 \cdot f(\langle child_1(n_3) \rangle) \\ &= 1 \cdot f(\langle child_0(n_7) \rangle) + 0 \cdot f(\langle child_1(n_7) \rangle) = 14 \end{aligned}$$

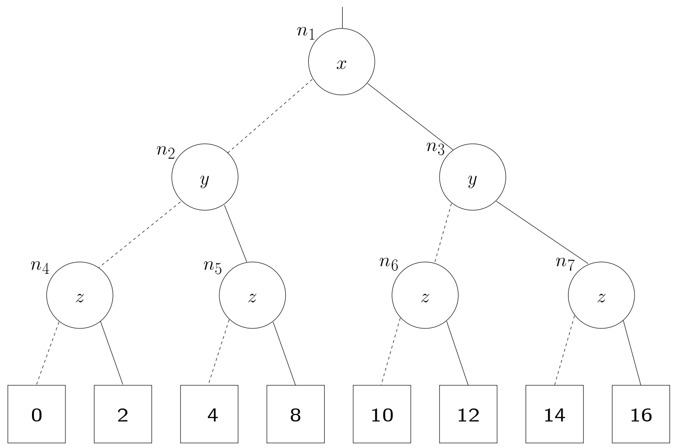

Figure 5-1: MTBDD representing the function $g(x,y,z) = -2xyz + 10x + 2yz + 4y + 2z$.

The MTBDD representation of the function g in Example 5-5 is exponential in the number variables, since each assignment of the variables result in a different function value. Consider instead the function $h = 3xyz$, which has the MTBDD representation given in Figure 5-2. This function illustrates the strengths of MTBDDs for representing function for which different variable assignments share function values.

Due to the potentially compact representation of functions, MTBDDs have been adopted to a wide variety of application domains such as symbolic model checking of probabilistic systems [KNPS99] and verification of arithmetic circuits [CFZ95a].

5.2 Motivation

As stated previously multi point linkage analysis is a computationally difficult task. That is, the running times of current algorithms grow exponentially in the number of non-founder in the pedigree and linearly in the number of markers. Furthermore, the space requirements also grow exponentially.

One of the main problems of the multi point probability computation in linkage analysis is the representation of, and operations on, probability distributions over exponentially many

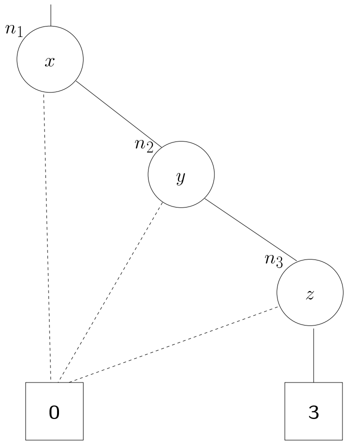

Figure 5-2: MTBDD representing the function $h(x, y, z) = 3xyz$.

inheritance vectors. Attempts such as founder reduction, [Gud00, KL98, KDRDL96], and founder couple reduction, [Gud00], have been introduced to decrease the number of inheritance vectors that need representation. This is a way of exploiting symmetries in probabilities by identifying inheritance vectors that always have the same probability, [CHI$^+$02, Gud00, KL98]. These methods lack the ability to take advantage of representing inheritance vectors with identical probability compactly, if the equality is not preserved across all markers.

Recently, decision diagram (DD) data structures have proven successful in exploiting symmetries for compact symbolic representation of large[4] state spaces, [CFZ95a, MBS95, BCM$^+$92]. Moreover, recursive algorithms have been developed to perform operations on the symbolic data structures thereby exploiting the compactness of the representation.

Many of the DD data structures represent functions of the type $\mathbb{B}^n \to D$, where D is some domain like $\mathbb{B}, \mathbb{Q}, \mathbb{R}$, etc., [Bry86, TP96, FYM93, CB]. Identifying inheritance vectors with probabilities can obviously be described as a function with signature $\mathbb{B}^n \to [0, 1]$ which makes existing data structures easily applicable. The worst case running time of algorithms on DD data structures are no better than the fastest algorithms using explicit representation, but they have proven highly efficient in real-life examples. For this reason these algorithms are referred to as *heuristic-based* algorithms.

In the following sections we further justify the investigation of heuristic-based algorithms for single and multi point linkage analysis.

5.2.1 Zero Probabilities

When investigation outputs of single and multi point probability computation most markers seem to have a high percentage of probabilities which are zero, that is inheritance vectors which are incompatible with the given genotype information.

[4] Even *infinite* state spaces, [RRK01, LWPY99].

In this section we examine the percentage of zero probabilities for single point probability[5] computation at different markers in pedigrees of different size (number of individuals). All pedigrees are actual real life examples provided by deCODE Genetics.

Figure 5-3 displays the percentage of zero probabilities in pedigrees of different size. The mentioned size is after the pedigrees have been subjected to founder and founder couple reduction.

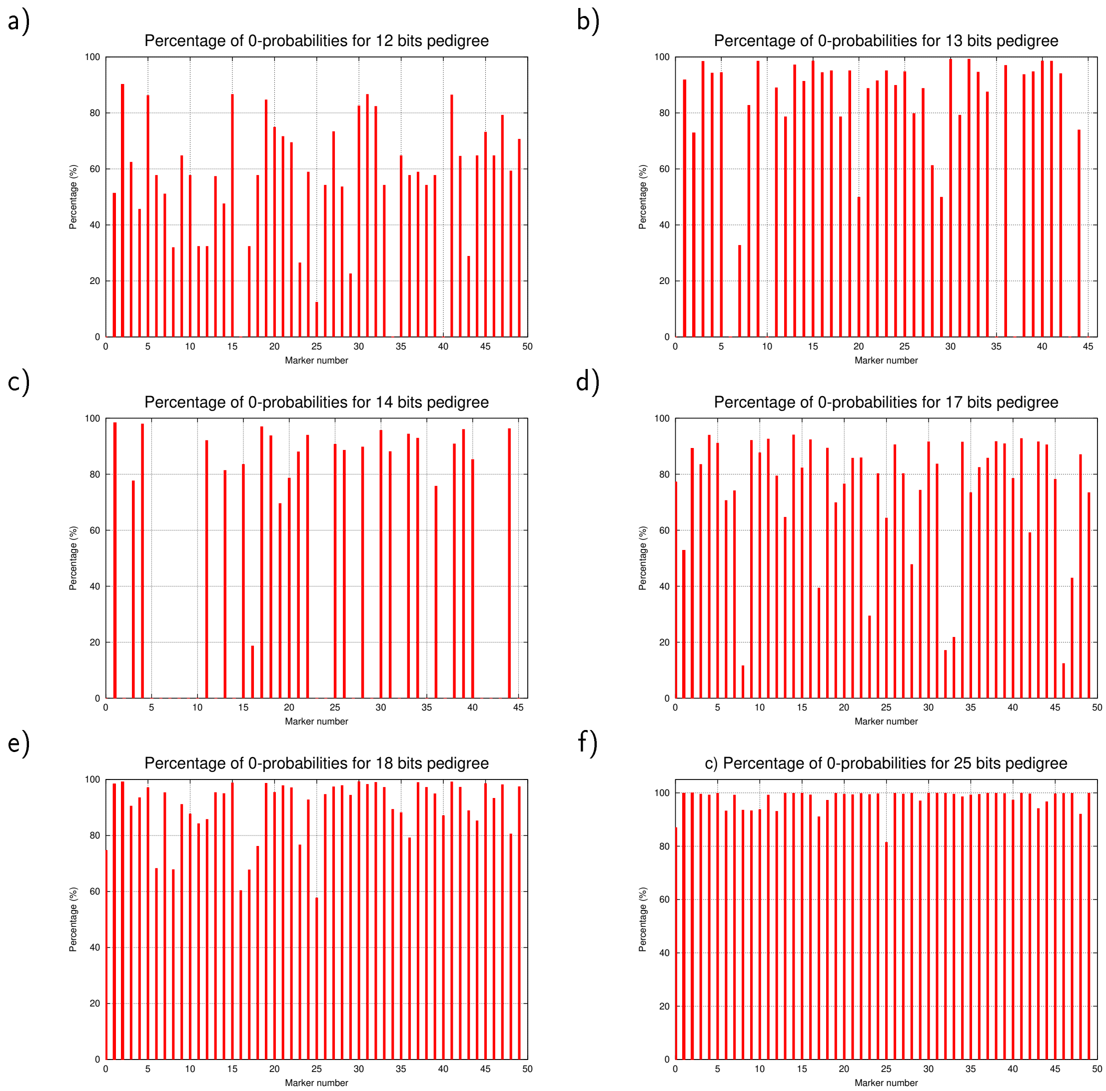

Figure 5-3: Percentages of zero probabilities at different markers for pedigrees of different size. a) 12 bits, b) 13 bits, c) 14 bits, d) 17 bits, e) 18 bits, and f) 24 bits. Founder and founder couple reduction have been performed on all pedigrees.

Figure 5-3 shows that for all pedigrees some markers have a high percentage of zero probabilities.

[5] From Equation 3-9 on page 28 we see that all probabilities which are zero in single point analysis are also zero in multi point analysis.

Some markers, such as marker 16 in Figure 5-3a, have no zero probabilities. This means that all inheritance vectors are compatible with the given genotype information, which in terms indicate that a low percentage of individuals (or even non at all) have been genotyped at this marker or it could be the case that most individuals have the same genotype. This is called a *highly uninformative marker* in contrast to *highly informative markers* where most inheritance vectors are incompatible. Uninformative markers add little or no information to the result of the analysis, since many or all inheritance patterns at the marker are possible given the observed data.

The graphs in Figure 5-3 show that a high percentage (some greater than 99.99%) of inheritance vectors are incompatible with the provided genotype information, which calls for a more compact representation of probability distributions. Furthermore, the tendency seems to be that larger pedigree have more informative markers than smaller pedigrees.

Since DD data structures have proven successful at representing symmetries, investigating their applicability in linkage analysis seems natural given the high percentage of zero probabilities.

5.3 Strategy

In this section we describe our strategy for testing the applicability of different DD data structures for representing inheritance vectors with corresponding probabilities. The parameters that will be tested are divided into two sub-categories, *memory usage* and *running time*.

We have chosen two measurements to estimate the memory usage of the data structures:

- **Final node count:** How many nodes are needed to represent single and multi point probabilities distributions.
- **Intermediate node count:** What is the maximum number of nodes needed when constructing single and multi point probabilities.

We measure the total running time of constructing a graph based on single and multi point probabilities on a common hardware platform.

5.4 Test Setup

In this section we describe the test setup - the hardware and the software - used in the experiments with the various data structures and variable orderings.

5.4.1 Overview

Allegro is the software currently used at deCODE Genetics (http://www.decode.is) to perform linkage analysis. Allegro has a fixed inheritance vector layout which corresponds to the order in which individuals are processed in the fast tree traversal algorithm, See [Gud00], [CHI$^+$02]. Allegro has a number of features irrelevant for this project, such as haplo-typing, QTL analysis, a number of different analysis models etc., and it is optimized for speed. These factors have made it difficult to modify the code for our needs - especially the fixed inheritance vector layouts and the code optimisations - so we have built our own software with the name *JavaAllegro*[6]. We have built JavaAllegro such that it reads and writes files in the same format as Allegro mainly for two reasons:

- Debug JavaAllegro by testing whether it produces the same output as Allegro when given the same input.
- Run the heuristic tests on realistic data, that we have been able to use the data obtained from deCODE Genetics without modifications.

We use a DD package called Technische Universität Darmstadt DD-package, or TUDD for short, [Hör97], which offers an API for construction and manipulation of a number of different DD-based data structures, such as binary decision diagrams, binary moment diagrams, hybrid decision diagrams, Kronecker multiplicative binary moment diagrams, etc[7]. We have build an application based on the TUDD package called *GraphBuilder* which reads the output files of Allegro and JavaAllegro and constructs one or more of the DD structures offered by the package. The output of GraphBuilder is statistical data, such as the number of nodes in a graph, running time, etc.

In the following we provide a more technical description of the software and hardware used.

Allegro

Allegro is written in C++ and compiles on POSIX-compliant operating systems. Allegro has a number of debug options, some which have enabled us to dump intermediate data, such as single point and multi point probabilities for all inheritance vectors for a pedigree, to disk. It is a command-line based tool which reads files in the LINKAGE input format, which consists of two files:

- **.pre-file** which contains one or more pedigrees and information, such as the relationships between individuals, how they have been genotyped, and their disease status.
- **.dat-file** which contains the genetic linkage map, hence information such as the recombination fractions, number of allelic states for each marker, and their allele frequencies.

The debug output we use (single point probabilities, left/right contributions, and multi point probability distributions) is in a simple text file format, where a line consists of three components 1) The marker number, 2) The inheritance vector in decimal form (base-10), and 3) The probability of that vector.

[6] The name "JavaAllegro" is misleading since it in no way is a Java implementation of all of Allegro's features, but only the single and multi point algorithms.

[7] We refer to Appendix B on page 117 for the definitions of the different data structures.

JavaAllegro

We have written JavaAllegro in Java (release 1.3.1) for fast proto-typing and flexibility. We have made it highly object orientated, which have enabled us to work with a number of different approaches for algorithms and inheritance vector layouts simultaneously. A number of different parameters enable the user to select which inheritance vector layout and which algorithms should be used.

The TUDD package

The TUDD package is written by Stefan Höreth as part of his Ph.D thesis. The TUDD package is written in C for efficiency, and it has its own built-in memory-manager, which enables it to cache certain operations for later reuse. The package can handle a number of different DD-based data structures: BDDs, FDDs, OKFDDs, KDDs, MTBDDs, EVBDDs, BMDs, *BMDs, p*BMDs, HDDs, *HDDs, p*HDDs, aHDDs, K*BMDs, pK*BMDs, ZDDs, and ZBDD. The package is convenient to use since all structures can be constructed and manipulated by the same set of operations.

The evaluation of the different DD-based data structures is of course very dependent on the implementation of the TUDD package. In the worst case: If the package contains bugs, the test data might be completely wrong. If the implementations of some of the DD-based data structures are more optimized than others, our results might be biased. From our experience, however, the TUDD package appears to 1) Provide correct results in terms of producing correct representations of functions, and 2) Perform as one would expect, meaning that the more complex the structure the longer the running times, e.g. building an EVBDD, where each edge has a value, takes longer to built than an MTBDD, where only the terminals have values.

We use version 0.8.4 of the TUDD package. For more information on the TUDD package see[8]:

`http://www.rs.e-technik.th-darmstadt.de/~sth/`

GraphBuilder

GraphBuilder is a small tool written to read the output of Allegro and JavaAllegro and produces a DD-based data structure. The user can via parameters instruct GraphBuilder to select any of the DD-based data structures provided by the TUDD package. Internally, the DD-based data structure is build inheritance vector by inheritance vector:

1. To begin with we have a constant diagram with one terminal "0". We call this diagram D_{total}. When all vectors have been processed this diagram will represent the entire probability distribution over inheritance vectors for one marker.

2. For each vector v read, we generate a formula based on v's binary string representation $s = b_1b_2b_3\dots b_n$, by creating a boolean variable x_i for every bit b_i. If the b_i is 0 we negate x_i otherwise we leave it.

[8] Last date checked: 26th April 2002.

3. All the literals are multiplied and their product is multiplied by the probability of the inheritance vector.

4. Now we have a diagram which represents the function with the value $P(v)$, if $x_1, x_2, .., x_n$ are assigned the binary string representation of v. For all other inputs the function is zero.

5. We add the newly created diagram to D_{total}, and proceed to step 2 with the next inheritance vector.

Example 35 (GraphBuilder: Creating a DD-based data structure for one inheritance vector) *Assume that we have an inheritance vector, v, with the binary string representation of $s = 0011$ and the probability of $P(v) = 0.3$. The function that represent v and its probability is:*

$$f(x_1, x_2, x_3, x_4) = \overline{x_1 x_2} x_3 x_3 \cdot 0.3.$$

TUDD then converts the function f into a diagram.

By summing the diagrams for all the inheritance vectors we obtain one diagram which represents the entire probability distribution.

Hardware

Timing experiments were performed on a Dual Pentium III coppermine 1 GHz with 1GB RAM running Solaris 8.

Summary

Our test setup consists of a number of applications which are all operated from a shell by command-line parameters. We have build a number of scripts to produce the various metrics and graphs for a combination of a DD-based data structures and input data (pedigrees). The scripts have made the experiments more efficient and less error-prone due to a large amount of automation.

The experimental results are very dependent on the implementation of the TUDD package, as mentioned, however, we have found no other package which provides as many different structures as TUDD. The fact that all structures are manipulated through the same set of generic operations suggests that the complexity is reflected by the running time, which has also been our experience from working with the package. The alternative would be to use a number different APIs for different structures, but then one would expect a much higher degree of diversity in terms of implementations.

5.5 Variable Ordering

It is a well-known fact that the size of DD-based data structures is often very sensitive to the variable ordering of the boolean functions which they represent. In this section we give a motivation and a formal description of six variable orderings which we use in the evaluation of the DD-based data structures. Since the size of DD-based data structures is usually very sensitive to the variable ordering, we go through the variable orderings in a formal and quite detailed manner.

All the DD-based data structures we work with require a fixed variable ordering, $x_1 \prec x_2 \prec \ldots \prec x_n$ such that a sub-tree of a node x_i only contains nodes x_j such that $j > i$. This means that every path form the root to a terminal respects the ordering. We would like to represent the probability distributions over inheritance vectors using DD-based data structures, and the set of variables VAR is the set of all meiosis $N \times \{p, m\}$. Therefore the variable ordering in our case is derived from the layout of inheritance vectors when they are written as elements of $\mathbb{Z}_2^n$ (or $\{p, m\}^n$).

Definition 13 on page 42 describes precisely the ordering of bits in a inheritance vector by an inheritance vector layout. For a given inheritance vector layout, o, we want the DD-based data structure to respect that layout. By respect we mean that if x_i represents meiosis (v, p) and x_j represents meiosis (v', m) and $o(v, p) < o(v', m)$ then $x_i \prec x_j$.

Example 36 (Inheritance Vector Layout) *Assume we have a pedigree with one non-founder* $child$*. We can have two different inheritance vector layouts for such a pedigree:*

$$\begin{aligned} layout1 &= ((child, p), (child, m)), \text{ or} \\ layout2 &= ((child, m), (child, p)). \end{aligned}$$

Depending on which layout is used the inheritance vector (p, m) *of course has two different interpretations. If* $layout1$ *is used* $v(child, p) = p$ *and* $v(child, m) = m$*, and if* $layout2$ *is used* $v(child, p) = m$ *and* $v(child, m) = p$*.*

A variable ordering in DD-based data structures representing probabilities over inheritance vectors is exactly the order of the bits in the inheritance vector layout. Hence, if $layout1$ from Example 36 is used, a DD-based data structure would have a node labeled with "$(child, p)$" as the root node, with two child-nodes labeled by "$(child, m)$". If $layout2$ is used it would be the other way around. See Figure 5-4 on the next page for a graphical representation of the two different variable orderings in a generic DD-based data structure.

In the next section we motivate and formalize different inheritance vector layouts, or variable orderings, which we evaluate for the various DD-based data structures.

5.5.1 The Six Variable Orderings

We will evaluate six different variable orderings, where three of the variable orderings are simply *mirrors* of the other three (the base orderings). Two of the base orderings use a top-down, breadth-first approach where members of a generation are placed adjacently in the inheritance vector layout, starting with the oldest generation. The two breadth-first base orderings are different in the respect that one use *grouping of paternal and maternal* bits while the other

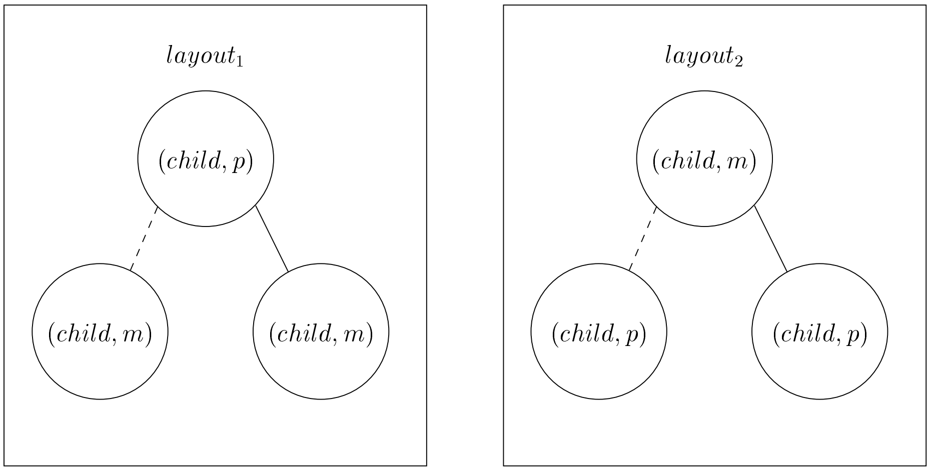

Figure 5-4: A graphical representation of a generic DD-based data structure with the two different variable orderings (inheritance vector layout) of Example 36.

does not. Grouping the maternal and the paternal bits means that all of the bits for the paternal alleles for siblings are placed consecutively followed by all of their maternal bits. For a motivation of why grouping paternal and maternal bits might help to reduce the size of a DD-based data structure see the following example:

Example 37 (Grouping bits) *Consider the pedigree shown in Figure 5-5 on the facing page. The children $C1$ and $C2$ have been genotyped with different allelic states (A/B and C/D), meaning that they could not have inherited from the same founder allele on neither the paternal nor the maternal side. In other words an inheritance vector, v, would be incompatible if either $v(C1,m) = v(C2,m)$ or $v(C1,p) = v(C2,p)$. If we have an inheritance vector layout, o, where no grouping is used, such as:*

$$
\begin{aligned}
o(C1,m) &= 1, \\
o(C1,p) &= 2, \\
o(C2,m) &= 3, \text{ and} \\
o(C2,p) &= 4,
\end{aligned}
$$

we would have to traverse three nodes labeled with "$(C1,m)$", "$(C1,p)$", and "$(C2,m)$" before we could exclude any inheritance vectors, since any value of $v(C1,m)$ and $v(C1,p)$ might lead to a compatible inheritance vector when $v(C2,m)$ and $v(C2,p)$ have not yet been evaluated.

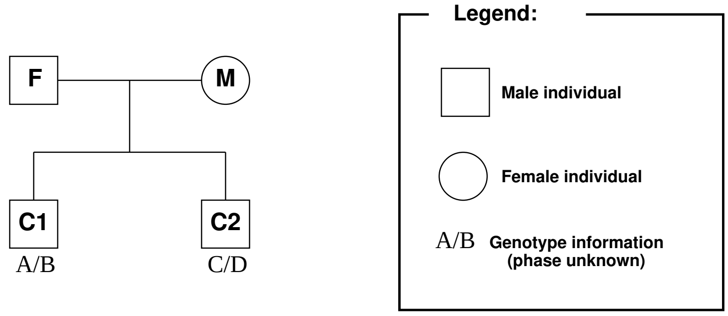

Figure 5-5: A pedigree with two founders and two non-founders. The non-founders $C1$ and $C2$ have been genotyped with four different allelic states A, B, C, and D. The pedigree is used in Example 37 to show why grouping bits might prove a good strategy for reducing the size of DD-based data structures.

If grouping is used and we assume we have following variable ordering:

$$\begin{aligned} o(C1, m) &= 1, \\ o(C2, m) &= 2, \\ o(C1, p) &= 3, \text{ and} \\ o(C2, p) &= 4, \end{aligned}$$

we will only have to traverse two nodes (the first labeled with "$(C1, m)$" and the other labeled with "$(C2, m)$") before some inheritance vectors can be excluded. It is natural to assume that the fewer nodes we have to traverse to exclude incompatible inheritance vectors the smaller the DD-based data structure.

A textual description of the three variable orderings is given below:

1. **Breadth first, no grouping (BFNG)**
2. **Breadth first, with grouping (BFWG)**
3. **Depth first (DF)**

The variable ordering BFNG corresponds to the order in which individuals are processed in the fast tree traversal algorithm, [Gud00] and [CHI+02].

Below we present the algorithms for the three base orderings.

BFNG-algorithm: Breadth First, No Grouping

The BFNG algorithm builds an ordered 1-indexed list of elements of the set $N \times \{p, m\}$ (the inheritance vector layout). The bijection of the inheritance vector layout, o, (See Definition 13 on page 42) then uses the list to determine the index of an allele. Before we can state the algorithm we need three functions: $children$, $sibs$, and DH (descendant height):

The children set is (obviously) the set of children for an individual:

Definition 19 (Children set) *Assume a pedigree $P = \langle F, N, father, mother\rangle$. Let $V = F \cup N$ be the set of individuals in P. The function $children : V \to 2^N$ is defined as:*

$$children(v) = \{\ n \in N \mid father(n) = v \text{ or } mother(n) = v\ \}, \tag{5-6}$$

where $v \in V$ is an individual in the pedigree.

The formal definition of the set of siblings *and* the set of half siblings is given below:

Definition 20 (Sibs) *Assume a pedigree $P = \langle F, N, father, mother\rangle$. Let N be the set of non-founders in P. The function $sibs : N \to 2^N$ is defined as:*

$$sibs(n) = \{\ x \in N \mid father(x) = father(n) \text{ or } mother(x) = mother(n)\ \}.$$

Note that the set of individuals returned by $sibs(n)$ contains n itself.

The descendant height, DH, is a recursively defined measure of how far an individual is from the most distant descendant in a pedigree. Hence, a measure of which generation an individual belongs to counting from the bottom. The formal definition is given below:

Definition 21 (Descendant Height) *Let $P = \langle F, N, father, mother\rangle$ be a pedigree with $V = F \cup N$. The descendant height $DH : V \to \mathbb{N}$ is given as:*

$$DH(v) \quad = \quad 1 + \max(\ \{\ DH(x) \mid x \in (children(v)\ \}\ ),$$

where $v \in V$.

See Example 38 for the $children$, $sibs$, and DH functions applied on a pedigree.

Example 38 (Children, Sibs, and Descendant Height function) *See Figure 5-6 on the next page. From the figure it is clear that $children(1) = \{4, 5\}$ while $children(14) = \emptyset$. The siblings set of individual 11 is $sibs(11) = \{11, 12, 13\}$.*

The descendant height of individual 4 is $DH(4) = 3$ since individual 8 has one child, individual 14, and $DH(14) = 1$ since 14 has no children. Note that two parents are only guaranteed to have equal descendant height when either one has only one mate in the pedigree.

Algorithm 5.5.1 on the facing page below, BUILDBFNG, builds an inheritance vector layout, an ordered list of pairs $N \times \{p, m\}$, which is used by the variable ordering function, o.

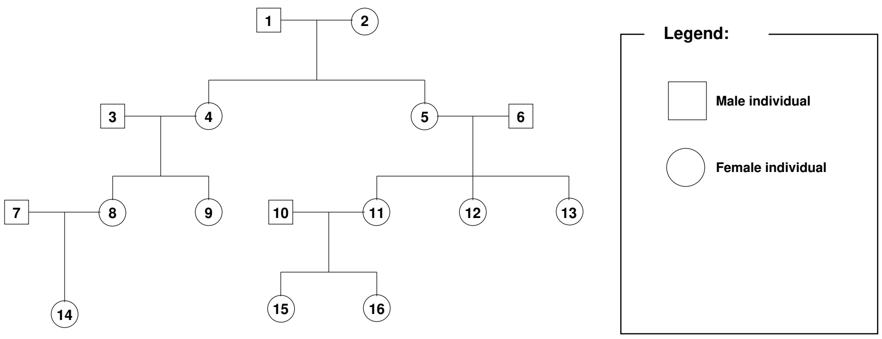

Figure 5-6: A pedigree with four generations. The pedigree is used in Example 38 on the preceding page to illustrate the $children$ set, the $sibs$ set, and the DH (descendant height) function.

Algorithm 5.5.1: $\text{BUILDBFNG}(P = \langle F, N, father, mother \rangle)$

$layout \leftarrow ()$ - Initialize an empty layout.
$todo \leftarrow N$ - $todo$ contains the individuals which still need to be added to $layout$
while $todo \neq \emptyset$
do
  $i \leftarrow BFNext(todo)$ - Find the individual with the greatest descendant height from $todo$.
  $S \leftarrow sibs(i)$ - Find all the sibs and half sibs to i.
  for $x \in S$
    do $\{layout \leftarrow layout :: (x, p) :: (x, m)$ - Add the paternal and the maternal bit of individual i.
  $todo \leftarrow todo \backslash S$ - Remove all the sibs from $todo$.
return $(layout)$

Note that it is often the case that a number of individuals have the same descendant height. In this case we assume that we have an initial ordering of the individuals in a pedigree such that the function $BFNext$ is deterministic. In our implementation, the initial ordering of individuals simply corresponds to the order in which individuals appear in the input file - hence, for a specific input $BuildBFNG$ always returns the same output.

BFWG-algorithm: Breadth First, With Grouping

Like in the BFNG-algorithm, the BFWG-algorithm (Algorithm 5.5.2 on the next page) finds the individual, i, with the greatest descendant height still not in the inheritance vector layout. Since we need to group paternal and maternal bits, we require two loops when the sibling set S is found: One for all the paternal bits and one for the maternal bits.

Algorithm 5.5.2: $\text{BUILDBFWG}(P = \langle F, N, father, mother\rangle)$

$layout \leftarrow ()$ - Initialize an empty layout.
$todo \leftarrow N$ - $todo$ contains the individuals which still need to be added to $layout$.
while $todo \neq \emptyset$
do $\begin{cases} i \leftarrow BFNext(todo) \text{ - Find the individual with the greatest descendant height from } todo. \\ S \leftarrow sibs(i) \text{ - Find all the sibs and half sibs to } i. \\ \textbf{for } x \in S \\ \quad \textbf{do } \{layout \leftarrow layout :: (x,p) \text{ - Add the paternal bit of individual } i \text{ to } layout. \\ \textbf{for } x \in S \\ \quad \textbf{do } \{layout \leftarrow layout :: (x,m) \text{ - Add the maternal bit of individual } i \text{ to } layout. \\ todo \leftarrow todo \backslash S \text{ - Remove all the sibs from } todo. \end{cases}$
return $(layout)$

DF-algorithm: Depth First

In human pedigrees the genotype information is often concentrated in the lower part of pedigrees, since the youngest generations are more likely to have been available for genotyping than older generations (See Section 3.1.2 on page 15 about missing and incomplete data). The motivation behind the depth-first approach is to use this tendency, by prioritizing paths from founders to leaf-nodes. By doing so, one could expect that incompatibilities arising from genotype information from *different* nuclear families are discovered in fewer steps compared to the breadth-first approach, where whole generations appear before any of their children.

The algorithm for building the depth-first inheritance vector layout is found in Algorithm 5.5.3.

Algorithm 5.5.3: $\text{BUILDDF}(P = \langle F, N, father, mother\rangle)$

global $layout$ - Will contain the inheritance vector layout when the algorithm terminates.
procedure $\text{ADDINDIVIDUALS}(todo)$
while $todo \neq \emptyset$
do $\begin{cases} i \leftarrow DFNext(todo) \text{ - Find the individual with the greatest descendant height from } todo. \\ layout \leftarrow layout :: (i,p) :: (i,m) \text{ - Add individual } i \text{ to } layout. \\ \text{ADDINDIVIDUALS}(children(i)) \text{ - Recurse one generation down.} \\ todo \leftarrow todo \backslash layout \text{ - Update } todo\text{, remove all individuals already added to } layout. \end{cases}$

main
$layout \leftarrow ()$ - Initialize an empty layout.
$\text{ADDINDIVIDUALS}(N)$ - Add all non-founders.
return $(layout)$

The Mirror Variable Orderings

The mirror variable orderings can be generated from the algorithms shown above simply by running the algorithms and reversing the resulting inheritance vector layouts. Hence, the first $N \times \{p, m\}$ pair of the original layout becomes from the last in the new layout, and so forth. The motivation behind doing so is that the all of the algorithms for the base orderings build the inheritance vector layouts in a top-down fashion. By mirroring the layouts we obtain variable orderings where bits are ordered in a bottom-up fashion. As stated above, genotype information in human pedigrees is often concentrated at the bottom, so using a bottom-up ordering might be more efficient than top-bottom, in term of the diagram size, since incompatible vectors can possibly be found by traversing fewer nodes.

5.5.2 Summary

We have presented a number of different inheritance vector layouts and their corresponding variable orderings. Of course there are many other orderings that we could try, however, we have picked those six orderings, since the ideas are quite different (BFNG is the default variable ordering, corresponding to the order in which individuals are processed in the single point analysis, in BFWG grouping of paternal and maternal bits is used, and in DF incompatibilities of genotypes in different nuclear families are presumably found in fewer steps). We expect that the results of using the six variable orderings given above will point in some direction so that the ideas behind one or more promising variable orderings can be combined and/or fine-tuned for one or more promising DD-based data structures. Another result of the evaluation of different variable orderings might be that the best ordering is dependent on the input.

Different variable orderings might be suited for different data structures, so each structure is tested with all of the variable orderings.

5.6 Analysis of DD Data Structures

In this section we present the performance test results of seven data structures: MTBDD, EVBDD, xHDD, BMD, *BMD, K*BMD. pK*BMD. The data structures are different in terms of the interpretation of the functions represented by nodes, and some data structures allow weights on edges - either additive, multiplicative, or both. For a listing of the properties of the seven data structures see Table 5-1 on the next page. For a more detailed description of the properties of the various data structures, including the formal definitions, examples, and references to the original articles, see Appendix B on page 117.

We could have gone through an evaluation of each data structure individually, node counts, execution time etc., however, as the results will show this would be pointless, since the results paint a very clear picture. Instead we first discuss the effect of the variable ordering on the size of the data structures, since the choice of variable ordering, quite surprisingly, has no significant influence on end diagram size. Then we discuss the performance of data structures using the *Shannon decomposition* compared to those using *positive* and/or *negative Davio decomposition*. Finally, we discuss the performance of data structures supporting different kinds of edge weights.

5.6.1 Test Data

DeCODE Genetics has supplied us with a large number of real pedigrees, but due to the company's legal obligations[9] we have been unable to obtain actual genotype information and

[9] Restrictions on the utilization of data in "Act on a Health Sector Database no. 139/1998" passed by the Icelandic government on December 17th 1998.

	S	pD	nD	Add. edges	Mul. edges
MTBDD	x				
EVBDD	x			x	
*HDD	x	x	x		x
BMD		x			
*BMD	x				x
K*BMD	x	x	x	x	x
pK*BMD	x	x	x	x	x

Table 5-1: Properties of the DD-structures used in the analysis. **S**, **pD**, and **nD** are short-hand for Shannon, positive Davio, and negative Davio decomposition, respectively. **Add. edges** is short-hand for additive edge weights and **Mul. edges** is short-hand for multiplicative edge weights.

actual genetic linkage maps. Instead we developed a tool called "GenoType" which reads a genetic linkage map and a pedigree and simulate Mendalian inheritance patterns, hence generates genotype information.

GenoType: An Application for Genotyping Individuals

We have used GenoType and a the genetic linkage map from one of the Allegro example pedigrees (F18). GenoType works in the following way:

Based on allele frequencies and a random function the founders are assigned allelic states. Then an inheritance vector is selected at random to be the true inheritance pattern for the first marker. Inheritance vectors are found by traversing the genetic linkage map from left to right, (assuming the first marker is the left-most marker). The outcome of each meiosis at a marker has a probability equal to the recombination fraction of being different from the same meiosis at the adjacent marker on the left-hand side. Hence, if the inheritance vector at marker M_i for the paternal allele of an individual n is $v(n,p) = p$, then the inheritance vector, w, at marker M_{i+1} has the probability of θ_i of being $w(n,p) = m$ for the meiosis. Based on the inheritance vectors and the genotyped founders the non-founders are assigned genotype information. To simulate that some individuals have been unavailable for genotyping the genotype information is deleted for some non-leaf individuals.

The Pedigrees and the Genotype Information

Out of the 650 pedigrees supplied by deCODE Genetics we have randomly selected 17 pedigrees, ranging from 4 to 20 bit (both inclusive) for our tests. We would like to have included larger pedigrees, however, due the space and time complexity of the algorithms based on simple array representations of probability distributions, we have not been able to do so. We have generated genotype information for the 17 selected pedigrees using GenoType as described above.

Table 5-2 on the next page shows the genotype statistics for pedigrees and the average diagram size for a MTBDD to represent the single- and multi point probability distributions. The numbers in the table indicate no obvious relationship neither between the fraction of genotyped individuals, heterozygotes, or homozygotes and the resulting MTBDD size.

	# ind.	# gt.	% gt.	# he.	% he.	# ho.	% ho.	S.P. n.c.	M.P. n.c.
4 bit	5	4	80%	2.8	56%	1.2	24%	3.2	4.4
5 bit	7	7	100%	5.6	80%	1.4	20%	4.2	7.8
6 bit	15	8	53%	5.8	37%	2.2	15%	8.8	37.4
7 bit	11	5	45%	4.0	36%	1.0	9%	22.4	69.0
8 bit	16	8	50%	6.4	40%	1.6	10%	18.6	74.2
9 bit	12	4	33%	2.8	23%	1.2	10%	20.0	133.4
10 bit	14	6	43%	4.2	30%	1.8	13%	55.0	349.8
11 bit	16	10	63%	6.0	38%	4.0	25%	84.0	375.4
12 bit	18	3	17%	2.4	13%	0.6	4%	23.0	301.8
13 bit	26	9	35%	8.2	32%	0.8	3%	77.0	270.2
14 bit	19	10	53%	8.6	45%	1.4	7%	239.0	1222.2
15 bit	21	9	43%	7.6	36%	1.4	7%	56.4	1328.8
16 bit	17	10	59%	8.8	52%	1.2	7%	258.4	675.0
17 bit	13	10	77%	7.6	58%	2.4	18%	72.0	168.0
18 bit	18	10	55%	7.8	43%	2.2	12%	96.4	1843.4
19 bit	26	9	35%	7.0	27%	2.0	8%	112.8	1850.4
20 bit	28	14	50%	11.6	41%	2.4	9%	330.8	27597.4

Table 5-2: Pedigree and genotype information. The table shows the number of individuals (**# ind.**), number of genotyped individuals (**# gt.**), percentage genotyped (**% gt.**), number of heterozygotes (**# he.**), percentage heterozygotes (**% he.**), number of homozygotes (**# ho.**), percentage homozygotes (**% ho**), the average node count for the MTBDDs representing the single point probability distributions (**S.P. n.c.**), and the average node count for the MTBDD representing the multi point probability distributions (**M.P. n.c.**). Each pedigree has genotype information for 5 markers, and the breadth first with grouping (BFWG) variable ordering was used in the MTBDD representation.

5.6.2 Variable Ordering

In Figure 5-7 on page 83 the average node counts for three data structures: MTBDD, BMD, and KxBMD using the six different variable orderings are plotted. As can be seen on the graphs, the choice of variable ordering has little effect on the resulting node counts. This is true for all of the seven data structures we have evaluated, independent of whether a data structure uses Shannon decomposition (like the MTBDD), or positive Davio (like the BMD), and whether edge weights are allowed (like the K*BMD). For some pedigrees one variable ordering is marginally better than others while the opposite might be true for other pedigrees (e.g. the BFNG ordering is better than the BFWG ordering for the 14 bit pedigree for multi point probability representation using MTBDDs, while for the 15 bit pedigree the BFWG ordering gives a lower node count than that of the BFNG ordering).

We believe the surprising result, that the variable ordering has only little effect on the resulting sizes of the DD-based data structures, is caused by the fact, that none of the data structures capture the structure of the domain. BMDs, for instance, are well suited for model checking of electronic circuits like multipliers since they utilize structure of the state space for those models, [BC94]. However, none of the data structures that we have evaluated appear to utilize the structure of the probability distributions. Still, some of the structures utilize the sparseness of the domain, e.g. for the 18 bit family, which require $2^{18} = 262.144$ values for each multi point distribution can be represented using an average of only 1.125 nodes per distribution using an K*BMD with Shannon decomposition (see Figure 5-7 on the next page).

5.6.3 Shannon Decomposition vs. Positive/Negative Davio Decomposition

The MTBDD and the EVBDD structures use Shannon decomposition while the BMD and *BMD use positive Davio decomposition. The remaining structures, the *HDD, the K*BMD, and the pK*BMD support all three decomposition types (one decomposition is specified for each variable, hence all nodes labeled with the same variable have the same decomposition type).

Recall from Section 5.1.1 on page 63 that we defined the Shannon decomposition of a function, g as:

$$g = \overline{x_i} g_{\overline{x_i}} + x_i g_{x_i},$$

where $g_{\overline{x_i}}$ and g_{x_i} are the negative and positive cofactors of g with respect to the variable, x_i. For MTBDDs we chose a given ordering of the variables and decomposed the function with respect to this ordering. Any node in an MTBDDs is then interpreted in terms of the Shannon decomposition. Other data structures are interpreted in terms of other decompositions, such as the positive or negative Davio decompositions. The positive Davio decomposition is defined as:

$$g = g_{\overline{x}_i} + x_i \cdot (g_{x_i} - g_{\overline{x}_i}).$$

For DD-based data structures this means that $child_0$ represents the negative cofactor and is always evaluated whereas $child_1$ represents $g_x - g_{\overline{x}}$ as is only evaluated when $x_i = 1$. The negative Davio decomposition is the "opposite" of positive Davio and is defined as:

$$g = g_{x_i} + (1 - x_i) \cdot (g_{\overline{x}_i} - g_{x_i}).$$

This means that for data structures interpreted in terms of negative Davio decomposition, $child_1$ is always evaluated and represents the positive cofactor whereas $child_0$ represents $g_{\overline{x}_i} - g_{x_i}$ as is only evaluated for $x_i = 0$. In Appendix B on page 117 we provide examples of the different data structures and their decomposition types.

Based on our test results we can safely conclude that positive and negative Davio decomposition do not work for representing probability distributions in linkage analysis. See Figure 5-8 on page 84 for a comparison of the node counts for the BMD and the MTBDD for representation of multi point probability distributions. As can be seen on the figure, the node count for the BMD is far higher than that of the MTBDD, for the 19 bit family - the largest family for which

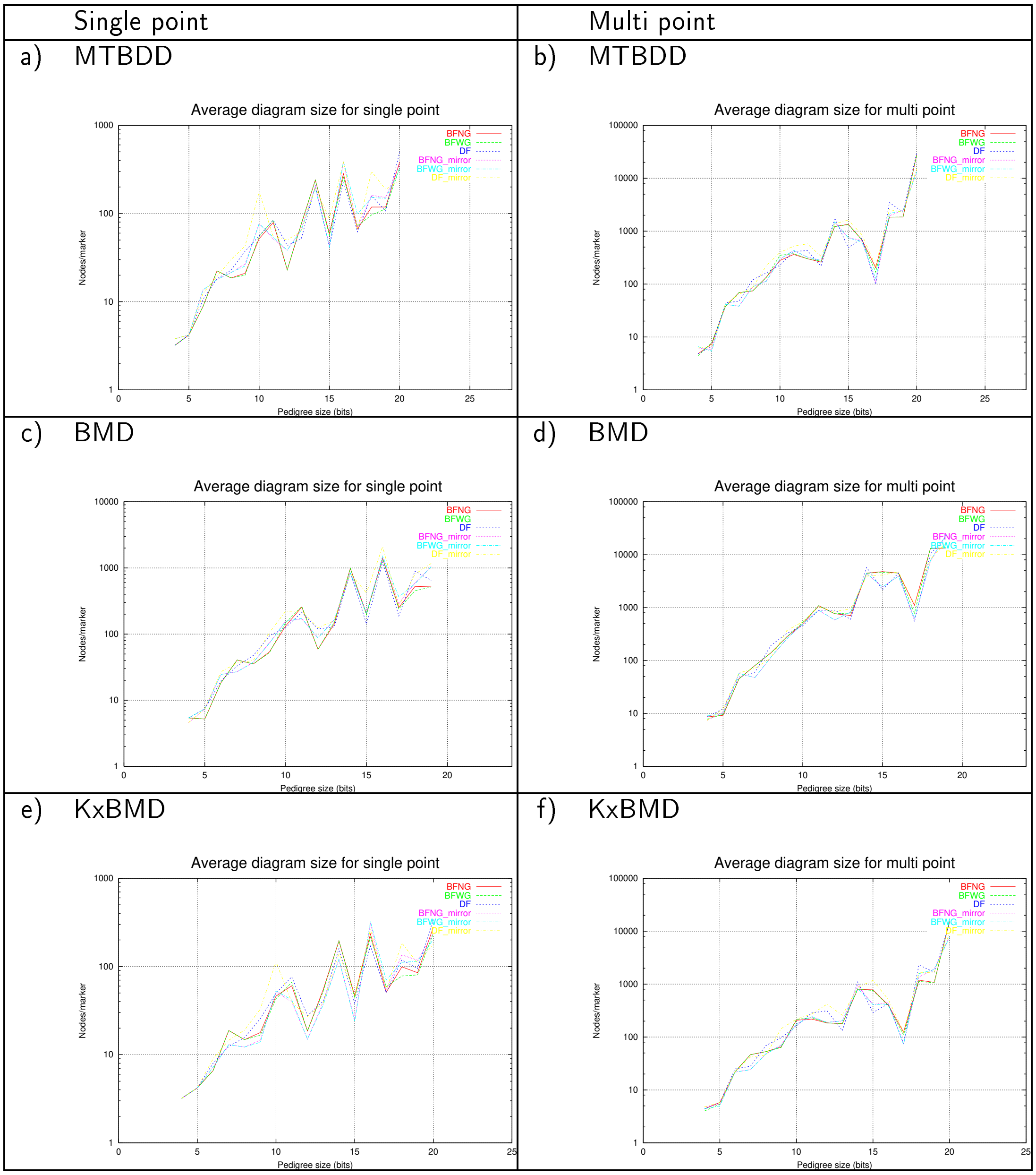

Figure 5-7: Node counts for three data structures, MTBDD a - single point, b - multi point, BMD c - single point, d - multi point, K*BMD e - single point, f - multi point. Note that the y-scale is a logarithmic scale, and that the BMD scales are different from the MTBDD and the K*BMD scales. See page 81 for a discussion of the graph.

we were able to build a BMD - requires an average node count of 13.548, while the average MTBDD size is only 1.850 nodes. We got similar results when we tested the *HDD, K*BMD, and pK*BMD with positive and negative Davio decomposition for every variable. Hence, we consider only Shannon decomposition in the remaining part of the report, so when we refer to the performance of the data structures supporting all three decomposition types, we mean their performance using Shannon decomposition only.

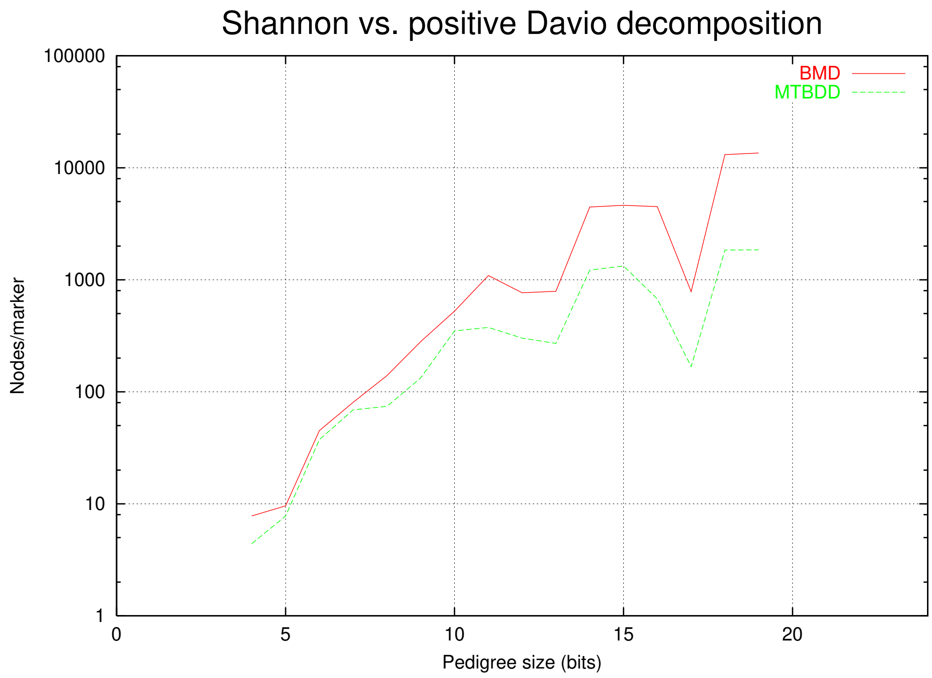

Figure 5-8: Comparison of the BMD and the MTBDD. Average node counts for multi point probability distributions with variable ordering BFWG.

5.6.4 The Effect of Edge Weights

The MTBDD does not allow edge weights, the EVBDD allows additive weights, the *HDD allows multiplicative weights, while the K*BMD and the pK*BMD allow both types of weights to be present on edges. Figure 5-9 on the next page shows a comparison between the node counts for data structures supporting different kinds of edge weights when representing multi point probability distributions. As it can be seen on the graph additive edge weights do not reduce the node counts considerably since the MTBDD and the EVBDD have almost identical node counts for all pedigrees. Multiplicative weights, on the other hand (supported by the *HDD and the K*BMD), reduce number of nodes required for representing single and multi point probability distributions. For the 16 bit pedigree the average node count for the MTBDD is 675 nodes per distribution while the average node count for the xHDD is 402 nodes per distribution. The combination of additive and multiplicative weights which is supported by the K*BMD is no better than just multiplicative weights - the *HDD and the K*BMD have almost identical node counts for all pedigrees as seen on Figure 5-9 on the facing page.

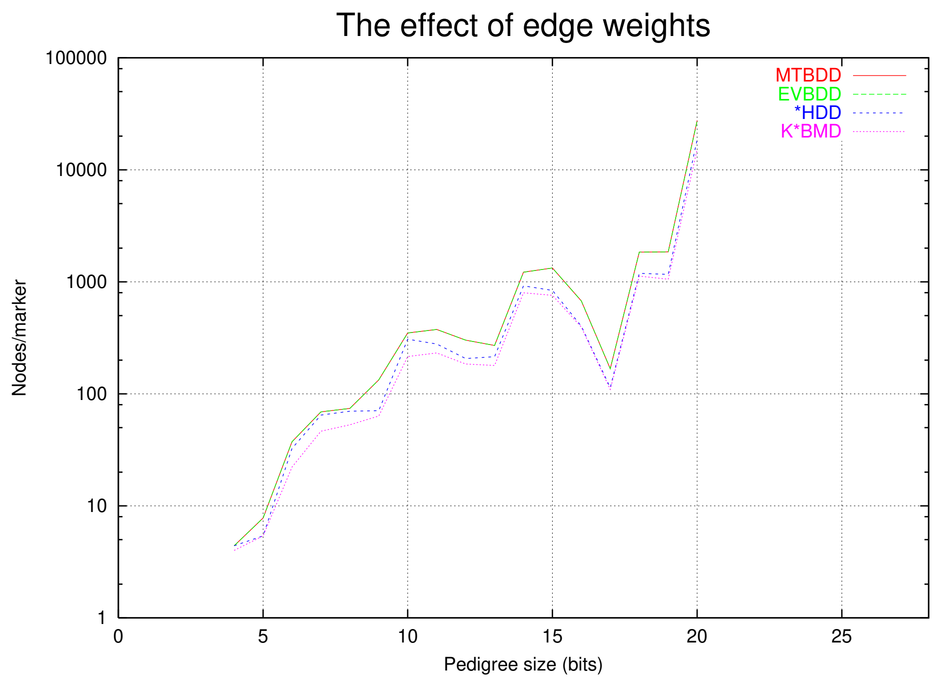

Figure 5-9: The effect of edge weights: MTBDD does not support edge weights, EVBDD supports additive edge weights, *HDD supports multiplicative, and the K*BMD supports both additive and multiplicative edge weights.

5.6.5 Maximum Node Count and Execution Time

The maximum node count refers to the maximum number of active nodes in the building process. For some problems like model checking, the intermediate node count might be many times greater than the final node count, [GV01]. We have to take this aspect into consideration when evaluating whether any of the DD-based data structures are suitable for use in linkage analysis. If the maximum node count turns out to be much higher than the final node count, DD-based data structures would not be feasible to use in linkage analysis due to the memory requirement and possibly long execution times. Figure 5-10 on the next page shows the average maximum and average final node count for MTBDDs representing multi point probability distributions. As it can be seen on the figure, the maximum node count is only fractionally higher than the final node count, and for the larger pedigrees it becomes insignificant.

5.6.6 Summary

In this section we have presented results to support the main conclusions of our tests:

- **Variable ordering:** No variable ordering is superior, the resulting node count for the DD-based data structures vary only little when using different variable orderings.

- **Decomposition types:** DD-based data structures based on Shannon decomposition to perform much better than those based on positive Davio (BMDs and *BMDs).

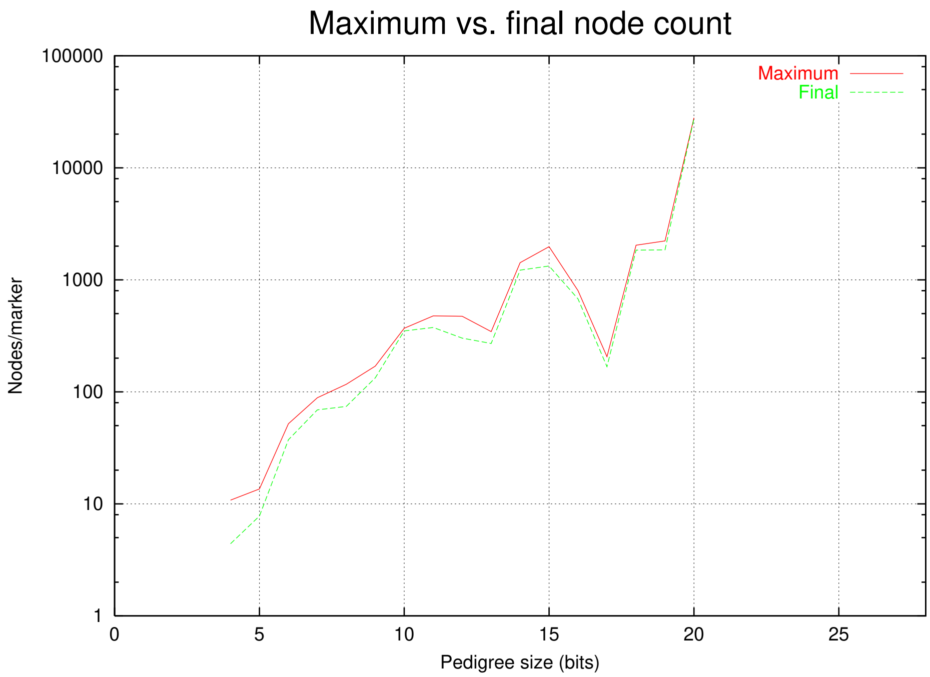

Figure 5-10: Maximum node count versus final node count for the MTBDD. The variable ordering BFWG was used.

- **Edge weights:** Additive edge weights do not minimize the representation, while the use of multiplicative weights gives some reductions in the number of nodes required for representation of probability distributions in linkage analysis.

5.7 Summary

We have presented an analysis of the performance of seven data structures which support different kinds of decompositions and some allow multiplicative and/or additive edge weights. The results of the tests show that Shannon decomposition-based data structures can in fact provide compact representations of probability distributions in linkage analysis. The MTBDD is the simplest Shannon decomposition-based data structure, since it does not allow any edge weights. The *HDD has a more compact representation in terms of node count, however, due to the multiplicative weights, the practical space requirements and the execution times show that the advantages of the *HDD might not be as great as the node counts show (See Figure 5-9 on the preceding page). Figure 5-11 on the next page shows the average execution time per marker for building *HDD and MTBDD data structures for representation of multi point probability distributions. As can be seen on the figure, the *HDD takes slightly longer to build than the MTBDD on average. This result is of course not surprising since operations on the *HDD are in practice more complex than operations on the MTBDD, due to the multiplicative weights.

Hence, the advantages of the *HDD due to its multiplicative weights in terms of reduced node counts compared to the MTBDD, is somewhat outweighed by its practical space requirements

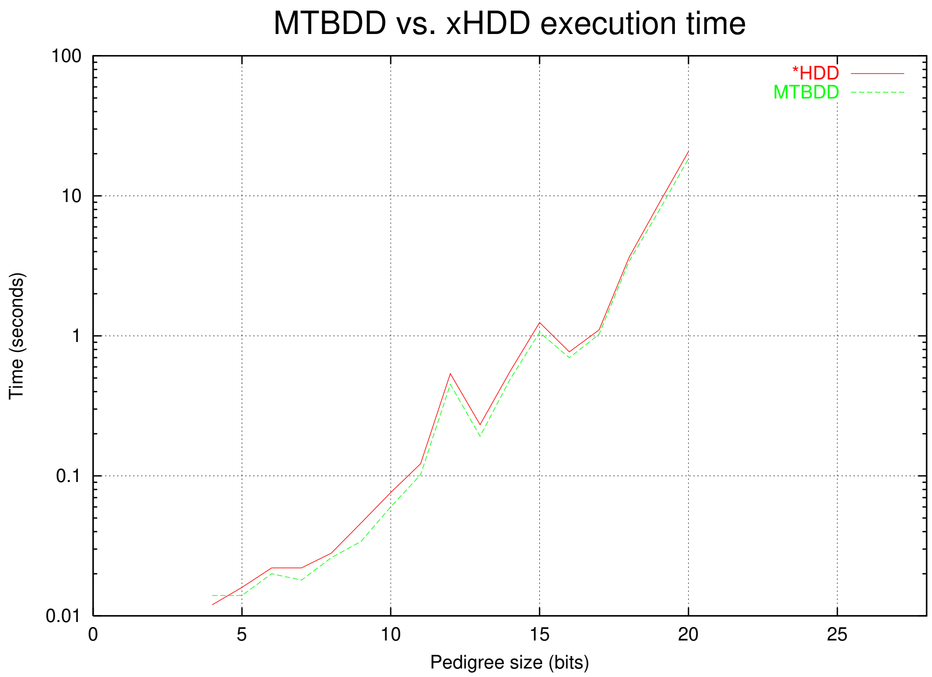

Figure 5-11: Comparison of the running times of building *HDD and MTBDD based representations of multi point probability distributions. The BFWG variable ordering was used.

and its longer execution times.

Heuristic-based Algorithms

CHAPTER 6

In this chapter we present two algorithms for performing the recombination transform in MTBDDs. One algorithm for performing the recombination transform on MTBDDs representing probability distributions over complete inheritance vectors, and one for distributions over founder reduced inheritance vectors. We show how dynamic programming techniques can be used in order to obtain an efficient algorithm. Finally, we present experimental results of an implementation of a pedigree analyzer based on MTBDDs and the recombination transform.

We have chosen the MTBDD data structure based on the evaluation of the performance of different DD-based data structures presented in the previous chapter. We could have chosen the *HDD, however, as discussed in the previous chapter, the apparent benefit of the *HDD in terms of lower node counts is somewhat outweighed by the additional storage requirement and practical complexity compared to the MTBDD.

6.1 Recombination Transform in Shannon-based DDs

In this section we show how the recombination transform presented in Section 4.1 on page 39 can be performed efficiently in MTBDDs. We first motivate our choice of transformation method, since there are at least three major methods for performing Kronecker product-based transforms.

6.1.1 Spectral Transforms of DD-based Data Structures

Several algorithms have been proposed for performing Kronecker product transforms on DD data structures, such as the Walsh transform and the Reed-Muller transform. We have considered three inherently different approaches for transforms on such structures:

- **By changing the decomposition of nodes:** For some transforms like the inverse Reed-Muller transform it suffices to change the interpretation (the decomposition type) of the nodes of a decision diagram representing the function, [CFZ95b, Rad].
- **By full transform:** A function, represented by a decision diagram, is transformed by defining a recursive transform operation on the DD data structure. The result of a full transform is a decision diagram representing the transform of the original function, [SK97].
- **By single coefficient transform:** Single spectral coefficients[1] are calculated recursively on the DD data structure. The result of single coefficient-based methods is simply the

[1] The result of multiplying a vector by a matrix (transform) is a vector. The elements in the resulting vector are referred to as *spectral coefficients* in the literature, [JSD01].

coefficient given as value and not a new diagram representing all spectral coefficients, as with the full transform, [JSD01].

The first method for spectral transformation of discrete functions, by changing the interpretations of nodes, works well for the inverse Reed-Muller transform, which has the base 2×2 matrix:

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

In [CFZ95b] it is shown that the MTBDD representation of the inverse Reed-Muller transform of a function is isomorphic to the Binary Moment Diagram[2] (BMD) of the function. Thus, the inverse Reed-Muller transform of a function represented by a BMD is simply obtained by interpreting the BMD (positive Davio decomposition) as a MTBDD (Shannon decomposition).

Unfortunately, there appears to be no feasible way to change the interpretation of nodes in a Shannon-based decision diagram to obtain its recombination transform. The reason there is a simple DD-based data structure which represent the inverse Reed-Muller transform is, that the base matrix has an entry with the value 0. Clearly, this is not the case for the base matrix of the recombination transform.

The second method, a full transform, can easily be implemented by a standard `Apply(...)` operation. However, the resulting decision diagram for representation of the left and right contribution vectors contain more nodes than those diagrams which represent the final multi point probability distributions. In Figure 6-1 on the next page the node counts of K*BMDs and MTBDDs representing left contribution vectors are plotted against the node counts of the final multi point distributions. This tendency is most likely du to the fact, that when a full transform is performed *all* the spectral coefficients have to be represented, and all of them have a non-zero value (assuming that $\theta > 0$). The multi point probability distribution, on the other hand, is often more sparse. For a marker all the inheritance vectors which had the probability of zero after the single point analysis[3] will also have zero probability after the multi point analysis[4], and as shown on Figure 5-3 in Section 5.2.1 on page 67 this can be a significant percentage especially for larger pedigrees. Thus, by performing a full transform, a large part of the coefficients might be computed in vain.

The third method for computing a Kronecker product-based transform of a function represented as a DD-based data structure, is by computing coefficients individually. The advantage of this approach is that we never build a structure for the entire transformed function, but we only have to compute coefficients for the non-zero entries of the distribution for which we are computing the left or right contributions. This might seem like a complex task to first identify all of the non-zero entries of the destination distribution, and then computing all of the necessary coefficients individually, but by applying a few tricks, we can actually obtain an efficient algorithm by this method. Notice, what is known as spectral coefficients within the theory of spectral transforms

[2] See Appendix B on page 117 for more on BMDs.
[3] Single point analysis: See Section 3.2.2 on page 21
[4] Multi point analysis: See (3-9) on page 28

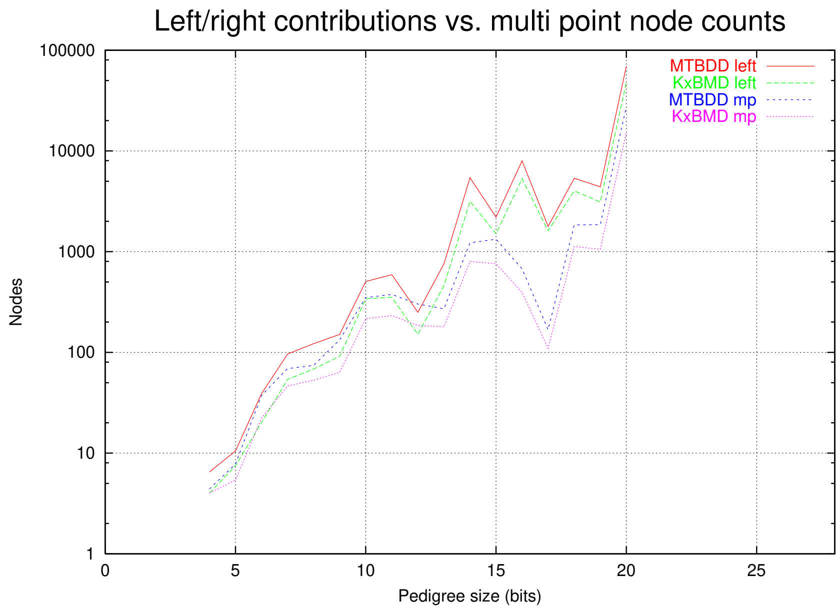

Figure 6-1: A comparison between the node counts for the K*BMDs and MTBDD left contribution vectors and the node counts for the final multi point distributions.

corresponds to contribution in linkage analysis. That is, retrieving some coefficient, $x \in \mathbb{Z}_2^n$, of the recombination transform, corresponds to the contribution from some adjacent marker to x. In the following we refer single spectral coefficients as single contributions.

In the next section we go through the algorithm, which we have developed, for the recombination transform on MTBDDs, e.g. how to compute a contribution for some inheritance vector from a probability distribution represented as a MTBDD. Afterwards, we show how to find and compute the necessary contributions in an order, which enables us to reuse intermediate results.

6.1.2 Single Contribution Recombination Transform

We first present the algorithm for the recombination transform on probability distributions over complete inheritance vectors (hence, without any founder reduction). Secondly, we show how this algorithm can be modified to work on founder reduced inheritance vectors[5].

We have chosen not to include the operation cache in the algorithms since it has no impact on the correctness proofs. Obviously, an operation cache is often necessary for algorithms using DD-based data structures, and later in this section we will explain how to reuse intermediate results to speed up performance.

Without founder reduction

The following algorithm for retrieving single contributions of the recombination transform is based on [JSD01]. In this papers single spectral coefficients are retrieved for the Walsh-

[5]See Section 4.2 on page 41 for more on founder reduction.

Hadamard transform of integer valued functions; however, it is noted in the paper that the method is applicable for all Kronecker-based transforms. Recall from Equation (4-1) on page 40 that the recombination transform can be given as a Kronecker product of 2×2-matrices. We extend the method to the recombination transform which, actually, simplifies the method of [JSD01].

We use MTBDD(g) to denote the MTBDD representation of some function g. The decomposition theorem of the recombination transform (Theorem 5 on page 40) states that the recombination transform $\mathcal{R}_n^\theta$ of some probability vector P can be computed recursively as:

$$C = \mathcal{R}_n^\theta \cdot P = \begin{bmatrix} (1-\theta)\cdot \mathcal{R}_{n-1}^\theta \cdot P_0 + \theta \cdot \mathcal{R}_{n-1}^\theta \cdot P_1 \\ \theta \cdot \mathcal{R}_{n-1}^\theta \cdot P_0 + (1-\theta)\cdot \mathcal{R}_{n-1}^\theta \cdot P_1 \end{bmatrix}, \tag{6-1}$$

where P_0 and P_1 are the negative and positive cofactors of P with respect to some variable. For a single contribution $C(v)$ to an inheritance vector, $v = (x_1, \ldots, x_n)$, (6-1) states that if $x_1 = 0$ in v then all inheritance vectors with first bit valued 0 are multiplied with $(1-\theta)$ (no recombination) and all inheritance vectors with first bit valued 1 are multiplied by θ (one recombination). Applying this reasoning recursively we get the correct contribution from all inheritance vectors to v. The algorithm which retrieves single contribution simply performs this operation on the MTBDD. Given the root node, m, of an MTBDD representing a probability distribution, g, over inheritance vectors, a recombination fraction θ, and an inheritance vector $v \in \mathbb{Z}_2^n$, Algorithm 6.1.2 computes the contribution to v from the inheritance vectors with probabilities given in $MTBDD(g)$. In the following we make no distinction between a function over $\mathbb{Z}_2^n$ and its representation as a naturally ordered vector of size 2^n, since any such function has a unique vector representation[6].

Algorithm 6.1.1: $\textsc{GetSingleContribution}(m \in V \cup T,\ \ (x_1, \ldots x_n) \in \mathbb{Z}_2^n,\ \ 0 < \theta \leq \frac{1}{2})$

if $m \in T$
  return $(label(m))$
$negChild \leftarrow \textsc{GetSingleContribution}(child_0(m), (x_1, \ldots x_n), \theta)$
$posChild \leftarrow \textsc{GetSingleContribution}(child_1(m), (x_1, \ldots x_n), \theta)$
if $x_{level(m)} = 0$
  return $((1-\theta)\cdot negChild + \theta \cdot posChild)$
if $x_{level(m)} = 1$
  return $(\theta \cdot negChild + (1-\theta)\cdot posChild)$

Theorem 8 (Single Contribution of Recombination Transform) *Given a function $g : \mathbb{Z}_2^n \to \mathbb{R}$ with naturally ordered vector representation, P, and a recombination fraction θ, then the contribution, $C(v)$, of $C = \mathcal{R}_n^\theta \cdot P$ to some $v = (x_1, \ldots, x_n) \in \mathbb{Z}_2^n$ from the elements in P is given as:*

$$C(v) = \textsc{GetSingleContribution}(m, (x_1, \ldots, x_n), \theta),$$

where m is the root node of MTBDD(g).

[6] Assuming that the vector is naturally ordered.

Proof: The proof is by induction on the number, n, of variables in g.
Basis step ($n = 1$):
The root node, m, of MTBDD(g) is either a terminal with value p or a node for which both children are terminals with values $label(child_0(m)) = p$ and $label(child_0(m)) = q$. If m is a terminal then the algorithm returns p. However, since $C(x_1 = 0) = (1-\theta)p + \theta p = C(x = 1) = \theta p + (1-\theta)p = p$ the algorithm is correct. If the root of $MTBDD(g)$ is a node, and we want to retrieve the contribution to (x_1) then the algorithm returns $(1-\theta)p + \theta q$ for $x_1 = 0$ and $\theta p + (1-\theta)q$ for $x_1 = 1$ and since:

$$\begin{bmatrix}(1-\theta) & \theta \\ \theta & (1-\theta)\end{bmatrix}\begin{bmatrix}p \\ q\end{bmatrix} = \begin{bmatrix}(1-\theta)p + \theta q \\ \theta p + (1-\theta)q\end{bmatrix} \quad \begin{matrix}(x_1 = 0) \\ (x_1 = 1)\end{matrix},$$

the base case holds.

Induction hypothesis ($n \leq i$):
We assume that for any function $g : \mathbb{Z}_2^j \to \mathbb{R}$ where $1 < j \leq i$ the algorithm returns the correct contribution to any $v' = (x_1, \ldots, x_j) \in \mathbb{Z}_2^j$ when given the root node of the MTBDD representation of g.
We now need to show that for the root node m of the MTBDD representation of any $g : \mathbb{Z}_2^{i+1} \to \mathbb{R}$ and any $v = (x_1, x_2, \ldots, x_{i+1}) \in \mathbb{Z}_2^n$, the algorithm returns the correct contribution to v. If m is terminal then g is the constant function $g = label(m)$ for which (6-1) states that the contribution to any v is $label(m)$ and the algorithm is correct.

If the root m has the label $label(m) = x_k$ for some $k > 1$ this means that the root node is not labeled with the variable, x_1 , since such a root, m', would have been redundant, hence $g_{\overline{x}_1} = g_{x_1}$[7] which implies that $child_0(m') = child_1(m')$. By the induction hypothesis the algorithm computes the correct contribution for some $w = (x_k, \ldots, x_{i+1})$. However, if $g_{\overline{x}_1} = g_{x_1}$ then (6-1) states that $C((0, x_2, \ldots, x_{i+1})) = C((1, x_2, \ldots, x_n)) = C'((x_2, \ldots, x_n))$ where C' is the contribution of a cofactor of g. This reasoning can be applied down to level k which gives $C(v) = C''(w)$, where $C''(w)$ is the contribution w for any cofactor function of g with respect to the variables $x_1, \ldots, x_{k-1}$.

If $label(m) = x_1$ we need to prove the correctness for both $x_1 = 0$ and $x_1 = 1$. We first choose $x_1 = 0$. (6-1) states that $C(v) = (1-\theta)C'(w) + \theta C''(w)$, where $w = (x_2, \ldots, x_{i+1})$ and C' and C'' are the contributions of the negative and positive cofactor functions of g. Since $child_0(m)$ and $child_1(m)$ represent $g_{\overline{x}_1}$ and g_{x_1}, respectively, the induction hypothesis states that $negChild = C'(w)$ and $posChild = C''(w)$ which implies that $C(v) = (1-\theta)negChild + \theta posChild$. Conversely, the algorithm holds for $x_1 = 1$. This completes the proof. □

Unlike the algorithm proposed in [JSD01], the recombination transform fully utilizes the compactness of the data structure. For performing Walsh-Hadamard transform on MTBDDs we

[7] Recall that $g_{\overline{x}_1}$ and g_{x_1} are the negative and positive cofactors of g with respect to x_1, that is, g_{x_1} is the function derived from g for which $x_1 = 1$ for all evaluations.

need to reintroduce every redundant node. This is not necessary for the recombination transform and we easily see why in (6-1). A node is redundant if and only if $P_0 = P_1$ for some variable, which means that the transform of such a node would is: $(1-\theta)P_0 + \theta P_1 = (1-\theta)P_0 + \theta P_0 = (1-\theta+\theta)P_0 = P_0 = P_1$.

The upper bound complexity of GETSINGLECONTRIBUTION is linear in size of the MTBDD since every node is visited once, that is $\mathcal{O}(|\text{MTBDD}(g)|)$, where $|\text{MTBDD}(g)|$ denotes the number of nodes in the MTBDD representing the probability distribution g.

With founder reduction

The straight-forward way to extend the algorithm to accommodate for founder reduction is to run GETSINGLECONTRIBUTION once for each of the 2^f hybrid recombination transforms, as described in Section 4.2 on page 41 (one transform for each of the possible $\langle q, I\rangle$ tuples). Obviously, this requires that we change the algorithm to allow both the recombination transform and the inverse recombination transform at a given level. Performing 2^f such transforms and multiplying each contribution for each transform with the appropriate scalar, and then summing the results, would compute a correct contribution. However, this would result in a complexity of $\mathcal{O}(2^f \cdot |\text{MTBDD}(g)|)$, where g is the probability distribution of founder reduced inheritance vectors. By using a different approach we can reduce the complexity to $\mathcal{O}(|\text{MTBDD}(g)|)$.

Recall from Section 4.2 on page 41 that we can rewrite the 2^f hybrid recombination transforms by using the distributive law of Kronecker product and componentwise addition on matrices. This transform, like the recombination transform[8], could be decomposed to achieve an complexity of $\mathcal{O}((n-f)2^{n-f})$, where n is the number of meiosis in the pedigree and f is the number of founders. So instead of running GETSINGLECONTRIBUTION 2^f times we construct an algorithm which follows the decomposition rules of the more efficient founder reduced recombination transform. However, this transform induced the restriction on the order of the bits in the inheritance vector that all meiosis concerning a given founder had to be grouped in sequence. Recall from the analysis of different data structures, See Figure 5-7 on page 83, that the variable ordering for the different data structures had virtually no impact on the size. Since MTBDDs representing single and multi point probabilities are insensitive to the variable orderings, inducing such restrictions can be easily justified.

Recall that the more efficient founder reduced recombination transform[9] used three sets, α, β, and Γ, to determine the context of any bit in the inheritance vector. A bit could either represent a meiosis unrelated to any founder (context 1, set Γ), it could represent the first bit in a sequence of bits concerning a meiosis of some founder (context 2, set α), or it could be in the middle of a sequence of bit concerning the meiosis of a founder (context 3, set β). The decomposition rule for each bit in the founder reduced inheritance vectors depends on the context of that bit. For this reason the algorithm needs to be able to handle each of the three cases and act according to the context of the variable for a given node in the MTBDD.

For readability we first give the algorithm for MTBDDs *with* redundant nodes. Afterwards we explain how to change the algorithm to allow redundant nodes to be absent from the MTBDD.

[8] See Definition 12 on page 39 for more on recombination transform.
[9] See page 60.

Algorithm 6.1.2: $\textsc{GetSingleFRContribution}(m \in V \cup T,\quad (x_1, \ldots x_n) \in \mathbb{Z}_2^n,\quad 0 < \theta \leq \frac{1}{2}, \alpha \in 2^{\mathbb{N}}, \beta \in 2^{\mathbb{N}}, \Gamma \in 2^{\mathbb{N}})$

if $(m \in T)$
  return $(\langle label(m), label(m) \rangle)$

$\langle negNormal, negReverse \rangle \leftarrow \textsc{GetSingleFRContribution}(child_0(m), (x_1, \ldots x_n), \theta, \alpha, \beta, \Gamma)$
$\langle posNormal, posReverse \rangle \leftarrow \textsc{GetSingleFRContribution}(child_1(m), (x_1, \ldots x_n), \theta, \alpha, \beta, \Gamma)$

if $(level(m) \in \Gamma)$ - $label(m)$ is in context 1 (not concerned with any founder meiosis)
  if $(x_{level(m)} = 0)$
    $res \leftarrow (1-\theta) \cdot negNormal + \theta \cdot posNormal$
    return $(\langle res, res \rangle)$
  if $(x_{level(m)} = 1)$
    $res \leftarrow \theta \cdot negNormal + (1-\theta) \cdot posNormal$
    return $(\langle res, res \rangle)$
if $(level(m) \in \alpha)$ - $label(m)$ is in context 2 (first bit in founder meiosis sequence)
  if $(x_{level(m)} = 0)$
    $res \leftarrow (1-\theta) \cdot ((1-\theta) \cdot negNormal + \theta \cdot posNormal) + \theta \cdot (\theta \cdot negReverse + (1-\theta) \cdot posReverse)$
    return $(\langle res, res \rangle)$
  if $(x_{level(m)} = 1)$
    $res \leftarrow (1-\theta) \cdot (\theta \cdot negNormal + (1-\theta) \cdot posNormal) + \theta \cdot ((1-\theta) \cdot negReverse + \theta \cdot posReverse)$
    return $(\langle res, res \rangle)$
if $(level(m) \in \beta)$ - $label(m)$ is in context 3 (bit is within a founder meiosis sequence)
  if $(x_{level(m)} = 0)$
    $normal \leftarrow (1-\theta) \cdot negNormal + \theta \cdot posNormal$
    $reverse \leftarrow \theta \cdot negReverse + (1-\theta) \cdot posReverse$
    return $(\langle real, inverse \rangle)$
  if $(x_{level(m)} = 1)$
    $normal \leftarrow \theta \cdot negNormal + (1-\theta) \cdot posNormal$
    $reverse \leftarrow (1-\theta) \cdot negReverse + \theta \cdot posReverse$
    return $(\langle normal, reverse \rangle)$

To prove the correctness of the algorithm we state an invariant for the recursion.

Theorem 9 (Invariant on $\textsc{GetSingleFRContribution}$) *Let g be any function $g : \mathbb{Z}_2^n \to \mathbb{R}$ stated as a vector P and it corresponding MTBDD[10] representation, MTBDD(g) and let α, β, and Γ be the index sets which determines the context of a bit in the inheritance vector ($\alpha \cap \beta = \alpha \cap \Gamma = \beta \cap \Gamma = \emptyset$). Let:*

$$\langle nor, rev \rangle = \textsc{GetSingleFRContribution}(m, x, \theta, \alpha, \beta, \Gamma),$$

[10] For the purpose of this invariant, redundant nodes are not removed from the graph.

be a call at some level of recursion, where $x = (x_1, \ldots, x_n)$ is the vector to which we want to compute the contribution, and $m \in V$ is any node in MTBDD(g). If P' is the cofactor function of P given by m then $\langle nor, rev \rangle$ satisfy the following invariant depending on the context of the variable, $label(m)$:

- ***Context 1:** the bit, $label(m)$, is unrelated to all founder meiosis ($level(m) \in \Gamma$):*

$$nor = rev = \begin{cases} (1-\theta)\Re^{\theta}_{n-level(m)}P'_0 + \theta \cdot \Re^{\theta}_{n-level(m)}P'_1 & \textit{if } x_{level(m)} = 0 \\ \theta\Re^{\theta}_{n-level(m)}P'_0 + (1-\theta) \cdot \Re^{\theta}_{n-level(m)}P'_1 & \textit{if } x_{level(m)} = 1 \end{cases}$$

- ***Context 2:** the bit, $label(m)$, is the first bit of a sequence concerning a founder meiosis ($level(m) \in \alpha$):*

$$nor = rev = \begin{cases} \begin{cases} (1-\theta)((1-\theta)\Re^{\theta,nor}_{n-level(m)}P'_0 + \theta\Re^{\theta,nor}_{n-level(m)}P'_1) + \\ \theta(\theta\Re^{\theta,rev}_{n-level(m)}P'_0 + (1-\theta)\Re^{\theta,rev}_{n-level(m)}P'_1) \quad \textit{if } x_{level(m)} = 0 \end{cases} \\ \begin{cases} (1-\theta)(\theta\Re^{\theta,nor}_{n-level(m)}P'_0 + (1-\theta)\Re^{\theta,nor}_{n-level(m)}P'_1) + \\ \theta((1-\theta)\Re^{\theta,rev}_{n-level(m)}P'_0 + \theta\Re^{\theta,rev}_{n-level(m)}P'_1) \quad \textit{if } x_{level(m)} = 1 \end{cases} \end{cases}$$

- ***Context 3:** the bit, $label(m)$, is within of a sequence of founder meiosis, but not the first in the sequence ($level(m) \in \beta$):*

$$\begin{aligned} nor &= \begin{cases} (1-\theta)\Re^{\theta,nor}_{n-level(m)}P'_0 + \theta\Re^{\theta,nor}_{n-level(m)}P'_1 & \textit{if } x_{level(m)} = 0 \\ \theta\Re^{\theta,nor}_{n-level(m)}P'_0 + (1-\theta)\Re^{\theta,nor}_{n-level(m)}P'_1 & \textit{if } x_{level(m)} = 1 \end{cases} \\ rev &= \begin{cases} \theta\Re^{\theta,rev}_{n-level(m)}P'_0 + (1-\theta)\Re^{\theta,rev}_{n-level(m)}P'_1 & \textit{if } x_{level(m)} = 0 \\ (1-\theta)\Re^{\theta,rev}_{n-level(m)}P'_0 + \theta\Re^{\theta,rev}_{n-level(m)}P'_1 & \textit{if } x_{level(m)} = 1 \end{cases} \end{aligned}$$

Basically, the invariant just guarantees that we always perform the right decomposition of a given node. That is, if the node is unrelated to all founder meiosis (context 1) we want to decompose with respect the normal recombination transform. If the node represents the first bit in a founder sequence (context 2) we need to decompose with respect to both the recombination transform and the inverse recombination transform and multiplying the appropriate factors before adding them. Finally, if the node represents a meiosis within a founder sequence (context 3) we need to perform the recombination transform and inverse recombination transform and keep them separate. The three different decomposition rules of the efficient founder reduced recombination transform stated on page 60 are precisely the three contexts captured in the invariant property. We now prove that the algorithm correctly identifies the three contexts and acts accordingly.

Proof: The proof is by induction in the number, n, of variables in g.
Basis case ($n = 1$):
The root node, m, of MTBDD(g) is a non-terminal with two, not necessarily distinct (since we assume the MTBDD to allow redundant nodes), children with $label(child_0(m)) = p$ and

$label(child_1(m)) = q$. We first prove if $label(m)$ has **context 1**, we chose $x_1 = 0$ ($x_1 = 1$ is proved similarly) and from the algorithm we get:

$$nor = rev = (1-\theta)\cdot negNormal + \theta \cdot posNormal,$$

but since m only has terminal children $negNormal = p$ and $posNormal = q$ and we easily verify that this is the normal recombination transform for $x_1 = 0$. $x_1 = 1$ is similar.
If $label(m)$ has **context 2**, we then need to decompose with respect to both the recombination transform and the reverse recombination transform. For $x_1 = 0$ we get:

$$res = (1-\theta)\cdot((1-\theta)\cdot negNormal + \theta\cdot posNormal) + \theta\cdot(\theta\cdot negReverse + (1-\theta)\cdot posReverse),$$

however since $negNormal = negReverse = p$ and $posNormal = posReverse = q$ we easily verify that this the correct decomposition with respect to context 2. And since $\langle res, res\rangle$ is returned the invariant holds. The case for $x_1 = 1$ is similar. If $label(m)$ has **context 3** we need to keep track of both the recombination transform and the reverse recombination transform and keep them separate. In case $x_1 = 0$ the algorithm gives:

$$\begin{aligned} normal &= (1-\theta)\cdot negNormal + \theta\cdot posNormal \\ reverse &= \theta\cdot negReverse + (1-\theta)\cdot posReverse \end{aligned}$$

Since $negNormal = negReverse = p$ and $posNormal = posReverse = q$ we can verify that $normal$ holds the normal recombination transform of g and $reverse$ holds the reverse recombination transform of g. Since the algorithm returns $\langle normal, reverse\rangle$ the invariant holds. The case for $x_1 = 1$ is similar. This concludes the basis case.

Induction Hypothesis ($n = i$):
We assume that according to all functions, g, with i variables, GetSingleFRContribution returns the appropriate tuple $\langle nor, rev\rangle$ for any contribution $x = (x_1, \ldots, x_i)$ with respect to the context of the root node of MTBDD(g).
We now need to prove that for any function over $i+1$ variables GetSingleFRContribution returns the appropriate tuple $\langle nor, rev\rangle$ for any contribution $x = (x_1, \ldots, x_{i+1})$, with respect to the context of the root node, m, of MTBDD(g), where P is the vector representation of g. Since redundant nodes are included in the MTBDD we know that $label(m)$ represents a variable with the context of x_1. We choose only to prove this for **context 2** since the proofs for the two other cases are similar. We first prove the case for $x_1 = 0$. We then know that $label(m)$ represents a meiosis which is the first in a founder sequence. The decomposition rule for the founder reduced recombination transform states for $x_1 = 0$ and context 2 the contribution is given as:

$$(1-\theta)\cdot((1-\theta)\cdot \Re_i^{\theta,nor} P_0 + \theta\cdot \Re_i^{\theta,nor} P_1) + \theta\cdot(\theta\cdot \Re_i^{\theta,rev} P_0 + (1-\theta)\cdot \Re_i^{\theta,rev} P_1),$$

and the algorithm returns:

$$res = (1-\theta)\cdot((1-\theta)\cdot negNormal + \theta\cdot posNormal) + \theta\cdot(\theta\cdot negReverse + (1-\theta)\cdot posReverse).$$

We know that $child_0(m)$ and $child_1(m)$ are the MTBDDs for P_0 and P_1, respectively. If the following bit, $label(child_0(m))$, has context 3, then by the induction hypothesis $negNormal = \Re_i^{\theta,nor} P_0$, $posNormal = \Re_i^{\theta,nor} P_1$, $negReverse = \Re_i^{\theta,rev} P_0$, and $posReverse = \Re_i^{\theta,rev} P_1$. It is easily verified that the invariant holds in this case. If $label(child_0(m))$ has context 1 or 2, then by the induction hypothesis $negNormal = negReverse = \Re_i^{\theta} P_0$ and $posNormal = posReverse = \Re_i^{\theta} P_1$, and in this case the founder reduced recombination transform states that $\Re_i^{\theta,nor} = \Re_i^{\theta,rev} = \Re_i^{\theta}$. The correctness of the invariant for context 2 is then apparent. If $x_1 = 1$ the proof is similar. Analogously, we could prove contexts 1 and 3. □

It directly follows from the invariant theorem of GETSINGLEFRCONTRIBUTION (Theorem 9) that for any founder reduced probability vector, $\widetilde{P}$, over $\mathbb{Z}_2^{n-f}$, with MTBDD representation MTBDD($\widetilde{P}$), we can compute any contribution $x = (x_1, \ldots, x_{n-f})$ as:

$$\text{GETSINGLEFRCONTRIBUTION}(m, x, \theta, \alpha, \beta, \Gamma), \tag{6-2}$$

where m is the root node of MTBDD($\widetilde{P}$), θ is the recombination fraction, and Γ, α, β are the index sets of bits with contexts 1, 2 and 3, respectively.

Since Algorithm 6.1.2 returns the correct contribution we can investigate the complexity of retrieving the contribution to a single inheritance vector. It is obvious that the algorithm visits every node in the MTBDD exactly once[11] and the algorithm is thus linear in the size of the MTBDD. That is, for a function g which assigns probabilities to inheritance vectors the complexity of retrieving the contribution to some inheritance vector is $\mathcal{O}(|\text{MTBDD}(g)|)$, where $|\text{MTBDD}(g)|$ is the size of the MTBDD.

GetSingleFRContribution on reduced MTBDDs

So far we have assumed that the MTBDD contains redundant nodes, however, we would like to compute the contribution based on fully reduced MTBDDs. In a reduced MTBDD(g), where $g : \mathbb{Z}_2^n \to \mathbb{R}$ we have to handle cases for a node, m, where $level(child_0(m)) \neq level(m) + 1$ and/or $level(child_1(m)) \neq level(m) + 1$. We will now explain how to do so based on the partial MTBDD depicted in Figure 6-2. Notice we have a number of redundant nodes shown to the left in the non-reduced MTBDD compared to the reduced MTBDD showed to the right. Assume that we want to compute the contribution for the node, m, labeled by x_i in the reduced MTBDD. We have that $level(child_0(m)) = k$ for some $k > i+1$, and we have to compute the same result as we would have done in the non-reduced case. Since the redundant nodes have been omitted, we have to compensate for their absence. How we have to compensate depends on the context of the redundant node, hence whether it is labeled with a bit in either Γ, α, or β.

Redundant nodes labeled with a bit in Γ (context 1): For nodes labeled with bits in Γ, we only have to perform the normal recombination transform. Since the node is redundant we have that $negNormal = posNormal$, hence the result of a normal recombination transform is either:

[11] We assume there is an operation cache such that if there is sharing of subgraphs, each subgraph is only traversed once.

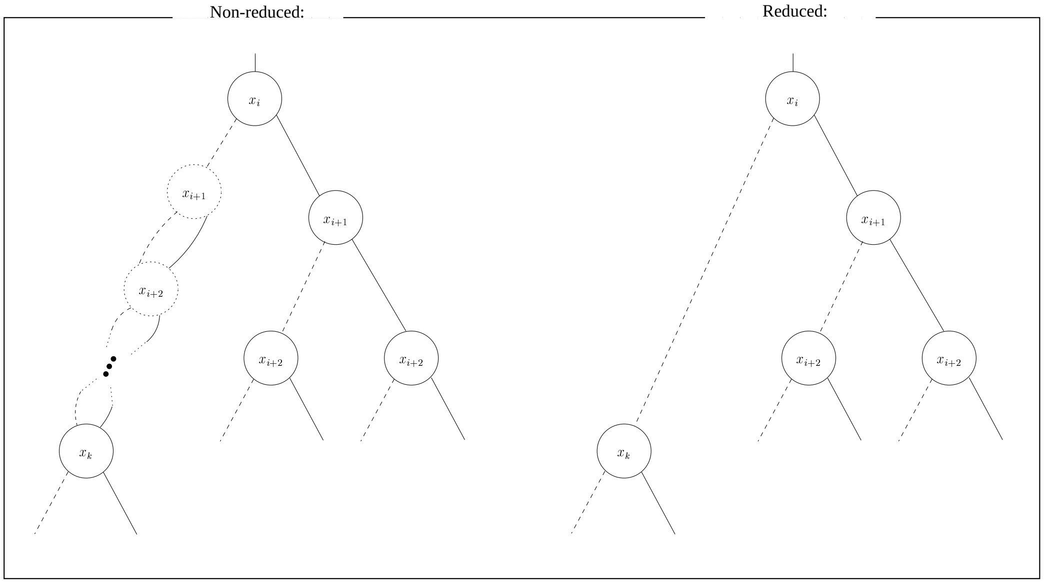

Figure 6-2: In the MTBDD shown to the left we have a number of redundant nodes which are absent in the reduced MTBDD.

$$res = (1-\theta)negNormal + \theta posNormal = negNormal,$$

or:

$$res = \theta negNormal + (1-\theta)posNormal = negNormal.$$

In both cases we can omit such nodes, since the result of calling GETSINGLEFRCONTRIBUTION on such a node is obtained by calling GETSINGLEFRCONTRIBUTION on its child (the node is redundant, thus $child_0(m) = child_1(m)$ if and only if m is redundant). Hence, we can ignore redundant nodes labeled with a bit in Γ, since the function would return a tuple identical to that received from $child_0(m)$.

Redundant nodes labeled with a bit in β (context 3): For nodes labeled with bits in β we have to perform the normal recombination transform and the reversed recombination transform. Since the node is redundant we have that $child_0(m) = child_1(m)$, and it follows that:

$$\langle negNormal, negReverse\rangle = \langle posNormal, posReverse\rangle.$$

Similarly to the previous case, we have that either:

$$normal = (1-\theta)negNormal + \theta posNormal = negNormal,$$

or:

$$normal = \theta negNormal + (1-\theta)posNormal = negNormal,$$

for the normal recombination transform, and:

$$reverse = (1-\theta)negReverse + \theta posReverse = negReverse,$$

or:

$$reverse = \theta negReverse + (1-\theta)posReverse = negReverse,$$

for the reverse transform. Hence, we can ignore nodes labeled by bits in β, since if we receive $\langle negNormal, negReverse\rangle$ from $child_0(m)$ (or $child_0(m)$) the algorithm returns $\langle normal, reverse\rangle$ where $\langle negNormal, negReverse\rangle = \langle normal, reverse\rangle$.

Redundant nodes labeled with a bit in α (context 2): For nodes, m, labeled with bits in α we have to be more careful, since the purpose of such a node is to combine the results of the normal and reversed recombination transforms of the child ($child_0(m) = child_1(m)$). If we receive $\langle normal, reverse\rangle$ from the child the algorithm returns:

$$\begin{aligned} res &= (1-\theta)\cdot((1-\theta)\cdot normal + \theta\cdot normal) + \theta\cdot(\theta\cdot reverse + (1-\theta)\cdot reverse) \\ &= (1-\theta)\cdot normal + \theta\cdot reverse, \end{aligned}$$

and we are forced to insert a virtual node which performs this gathering of $normal$ and $reverse$. However, in the case where $normal = reverse$ the algorithm returns $\langle normal, normal\rangle$, and we need not insert a virtual node.

Figure 6-3 summarizes the result of calling GetSingleFRContribution on redundant nodes. Thus, for GetSingleFRContribution to be applicable to reduced MTBDDs we simply have to keep track of whether some nodes, m, with $level(m) \in \alpha$ would have been present, had the MTBDD been non-reduced, (See Figure 6-2), and introduce a virtual node at that level, if and only if this is the case.

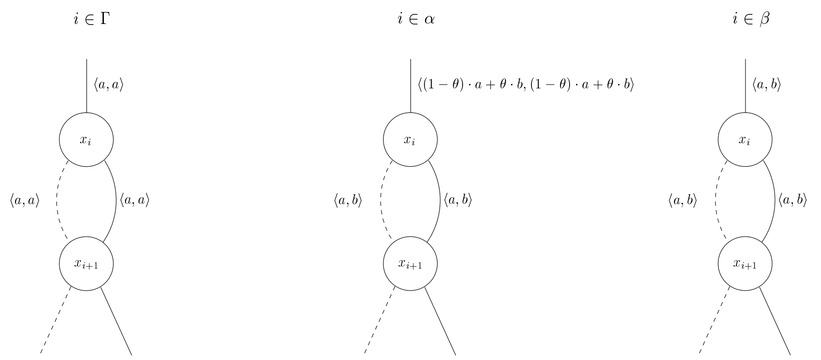

Figure 6-3: Summary of the result of calling GetSingleFRContribution on redundant nodes. It is only in the case that a redundant node is labeled by a bit in α that we have to introduce virtual nodes. A label on an edge going into a node refers to what is returned by GetSingleFRContribution when called on the node.

6.1.3 Reusing Intermediate Results

The complexity of computing the contribution to one inheritance vector, $v = (x_1, \dots, x_n)$, is linear in the number of nodes in the MTBDD, since we have to visit every node once and compute its transform based on the inheritance vector, v. However, since the result of a transform of nodes on a level, i, for some x_i is only dependent on the value of the bits $x_i, \dots, x_n$, we can save time by applying dynamic programming techniques. We use dynamic programming by storing the transform of each node between calls to GETSINGLEFRCONTRIBUTION for different inheritance vectors.

When GETSINGLEFRCONTRIBUTION is called on a new v' after computation of the contribution to v, we only have to compute the transform of those nodes dependent on the value of the bits for which the prefix of v and v' differ. Hence, if for $v = (x_1, \dots, x_n)$ and $v' = (x'_1, \dots, x'_n)$, we have that $x_i = x'_i$ for all i greater than some j we can reuse the transformations of the nodes, $m \in V$, for which $level(m) > j$. When GETSINGLEFRCONTRIBUTION is called with a vector v' we just have to check whether we already computed the transform for some postfix of v', if so we can return the previously computed result without any further recursion and compute the transform for the prefix. In DD terms this is known as an *operation cache*.

Using an operation cache we only have to transform a level in a MTBDD at most $2^{n-level(m)+1}$ times, where n is the length of a reduced inheritance vector. Hence, we would only have to transform level n, the lowest level in an MTBDD) at most twice, by computing the contributions for all of the inheritance vectors for which the n-th bit is 0, and then for the vectors for which $x_n = 1$. We would only have to transform the nodes at level $n-1$ four times, since we first compute all the contributions for vectors for which $v = (\dots, 0, 0)$, then $v = (\dots, 1, 0)$, then $v = (\dots, 0, 1)$, and finally $v = (\dots, 1, 1)$. Similarly we have to transform nodes at level $n-2$ at most eight times etc. We know that each call to GETSINGLEFRCONTRIBUTION is linear in the size of the MTBDD. Then by using an operation cache, the complexity of retrieving all 2^{n-f} contributions from a MTBDD of size |MTBDD| becomes $\mathcal{O}(2^{n-f} \cdot \log|\text{MTBDD}|)$. As a result of this, if the MTBDD is exponential in size we get $\mathcal{O}((n-f) \cdot 2^{n-f})$, which the same complexity as current algorithms which use fast Fourier transform. However, in practice the MTBDDs are often not exponential in size and we do not need to compute all 2^{n-f} contributions, since there are usually a fraction of the inheritance vectors, at each marker, with zero-probabilities.

We can actually come up with a good strategy of finding the inheritance vectors with a non-zero probabilities in a distribution such that we do not need the operation cache. Instead we only store *one* result per node. This requires that we somehow have find inheritance vectors with non-zero probabilities which share as many operations as possible consecutively. Unfortunately, there appear to be no easy way of doing so if the variable ordering is the same for the MTBDDs which we are computing the contributions from and to.

Recall from Section 5.6.2 on page 81 that the sizes of the DD data structures representing probability distributions are practically insensitive to the variable ordering. Due to this observation we can permit ourselves to use *different* variable orderings at each marker without any penalty in terms of memory usage. If we use the BFWG-variable ordering for every other marker and BFWG-mirror for the remaining markers, we can easily find inheritance vectors with non-zero probabilities in the optimal order. Recall, that BFWG stands for Breadth-First With Grouping and that its mirror ordering is exactly the ordering where the inheritance vector layout has been mirrored or reversed[12]. This means that the variable orderings for two adjacent markers M_i and M_{i+1} are opposite in the sense that the label of the root node for the MTBDD for M_i is the label of the nodes at level n, of the MTBDD for marker M_{i+1}, and vice versa. Due to this relationship between the variable orderings, we simply have to do a depth-first scan for inheritance vectors with non-zero probabilities in the MTBDD representing the probability distribution, to which we are computing the contribution, in order to reuse intermediate results optimally.

To clarify this, we saw above that the contributions for two inheritance vectors $v = (0, x_2, \dots, x_n)$ and $v' = (1, x_2, \dots, x_n)$ share operations for all nodes in the MTBDD except for the top level. However, when traversing some MTBDD to determine which contributions to compute, v and v' are located on different branches in the MTBDD with respect to the root node. That is, if v has integer interpretation x then v' has interpretation $2^{n-1} + x$. If instead the variable ordering is interchangeably switched between markers, such that $v = (x_n, \dots, x_2, 0)$ and $v' = (x_n, \dots, x_2, 1)$ would have integer interpretations y and $y + 1$ and would be identified consecutively during a breath first search of the MTBDD.

6.2 Experimental Results

In this section we present the experimental results of an implementation of the founder reduced recombination transform on MTBDDs for multi point linkage analysis in JavaAllegro. We compare the multi point probability computation performance of JavaAllegro and Allegro.

We present the test setup, some notes on the implementation, the results, and finally an analysis of the results.

6.2.1 Test Setup

The results presented were obtained from experiments conducted on a dual Pentium III 1GHz with 1GB of memory running Solaris 8. Allegro was compiled using `gcc` 2.95.3 and JavaAllegro was compiled using Sun's `javac` version 1.3.1-b24 and `Java HotSpot` virtual machine 1.4.0-b92.

6.2.2 Implementation Notes

We have extended JavaAllegro to use MTBDDs for representation of probability distributions and left and right contribution vectors. We have made our own specialized MTBDD package since we have been unable to find an existing fast and flexible package for Java, in which we could implement the founder reduced recombination transform. Our implementation uses the floating point data type `double` instead of `float` as Allegro does. The `float` is 4 bytes wide, while the `double` is 8 bytes wide. Variables of the type `float` can range from $1.401298 \cdot 10^{-45}$ to $3.402823 \cdot 10^{38}$, while variables of the type `double` can range from $4.940656 \cdot 10^{-324}$ to

[12] See Section 5.5.1 on page 73 for more on the different variable orderings.

$1.797693 \cdot 10^{308}$. We have decided to use `doubles` since they provide much better precision than `floats`, and their larger range becomes important as the size of pedigrees increase[13].

JavaAllegro is a prototype and it is written in Java. It has not undergone heavy optimizations like other industrial-strength packages, e.g. Allegro has been low-level optimized with loop-unrolls and CPU cache-optimizations, [Gud00]. However, since we measure the performance of two inherently different methods, explicit representation versus symbolic representation and the associated algorithms, the experimental results can still prove as a basis for comparison.

6.2.3 Results

The experimental results are shown in Figure 6-4 on the next page. The figure shows the running time of multi point probability computation of 5 markers. We chose only 5 markers to prevent Allegro from swapping during analysis. Note that the running time of Allegro does *not* include single point probability computations, haplotyping, pair-wise IBD estimation, or such.

Figure 6-4 shows two plots for JavaAllegro, one for perfect information, and one for partial information. *Perfect* and *partial* information refers to whether all or just some of the individuals in the pedigree have been genotyped, respectively.

6.2.4 Analysis

The analysis is divided into three parts: JavaAllegro compared to Allegro, JavaAllegro's performance on perfect information, and JavaAllegro's performance on partial information. There are two plots for Allegro's running time on Figure 6-4, one for the running time for pedigrees up to 25 bits, based on actual test runs, and one for larger pedigrees for which we have extrapolated Allegro's running time.

Allegro vs. JavaAllegro

Figure 6-4 on the following page shows that Allegro is somewhat faster than JavaAllegro for small pedigrees (< 22 bits), for the 21 bit pedigree Allegro uses 17 sec. while JavaAllegro uses 32 sec. For pedigrees with a size up to 17 bits Allegro actually uses less than 1 second, while JavaAllegro uses 6-7 seconds. The reason for this difference is most likely, that the Java virtual machine does garbage collection, and that for pedigrees with a size less than 22 bits the overhead of the MTBDD house-keeping exceeds the running time of the array based approach.

Due to the memory requirements of Allegro we were unable to compare running times for pedigrees larger than 25 bits, since Allegro required at least 1.6GB primary memory to analyze the 26 bit pedigree. However, given the explicit array representation of probability distributions and contribution vectors, which Allegro uses, its running time and memory usage are easily extrapolated. The running time is assumed to double every time pedigrees grow one bit in size.

The theoretical minimum memory usage of Allegro's approach is given by the following equation:

[13] Assume a pedigree with 25 founders, which gives 50 founder alleles. Assume a founder allele assignment for which all of the allelic states have a frequency of approximately 5%, hence, the founder allele assignment would have a probability of approximately $0.05^{50} = 8.9 \cdot 10^{-66}$ which is outside of the range offered by the `float`.

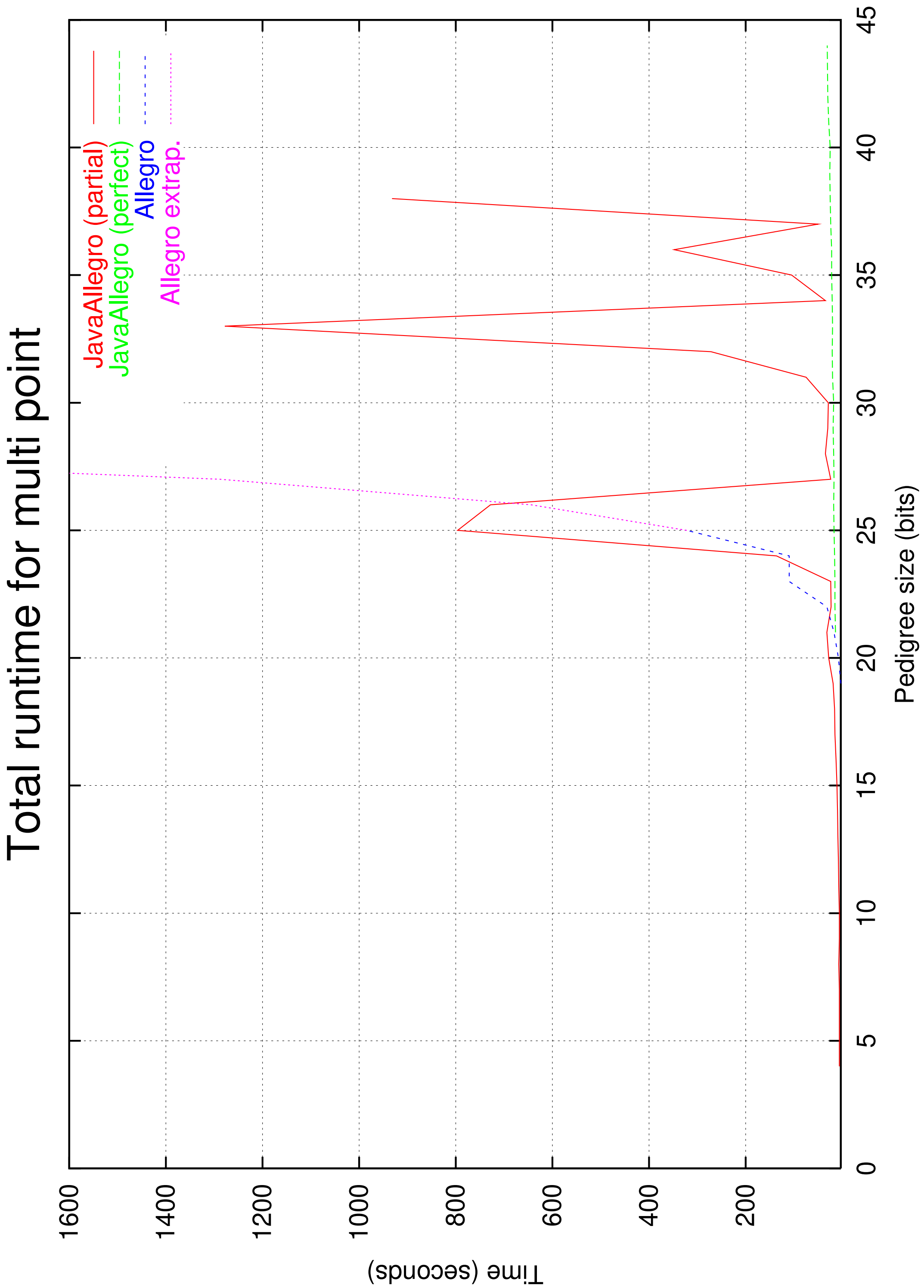

Figure 6-4: A comparison of Allegro and JavaAllegro on pedigrees of size 4 to 38 bits. The running time of Allegro has been extrapolated after 25 bits since the memory requirement exceeded the available primary memory. The term "perfect" refers to all individuals in the pedigree were genotyped, while "partial" means that approximately 50% of all non-leaf individuals were genotyped.

$$Allegro_{mem} = 2^{|P|} \cdot sizeof(\texttt{float}) \cdot 3m, \tag{6-3}$$

where $|P|$ denotes the size of the pedigree in bits after founder and founder couple reductions, m denotes the number of markers in the analysis. The size of one distribution/contribution vector is one probability per inheritance vector, hence $2^{|P|} \cdot sizeof(\texttt{float})$. And we need one single point distribution, one left contribution vector, and one right contribution vector per marker[14], hence the $3m$. Using (6-3) we can easily see that even with a 30 bit pedigree we would need more than 60GB of storage. For a 40 bit pedigree we would need more than 60TB for just five markers!

If the running time of Allegro is extrapolated in the same manner, wrongfully assuming that no swapping is required, we get a result similar to that of the storage requirements. The pink graph in Figure 6-4 illustrates the extrapolated running time of Allegro assuming that time doubles for every increment of pedigree size. If it is assumed that analyzing a 25 bit pedigree takes 322 seconds, as the result of our test show, analyzing a 30 bit family would take 2 hours and 50 minutes, while a 40 bit would take more than 122 days.

Notice that the running times of Allegro for the 23 and 24 bit pedigrees are almost identical. The reason for this is that Allegro uses founder couple reduction[15], and can reduce the pedigree size of the 24 bit pedigree to 23 bits, since there is an ungenotyped founder couple.

Contrary to Allegro, JavaAllegro was able to analyze every pedigree in the test, never exceeding the memory limit of 1GB. For some of the pedigrees with partial genotype information, the running time matches that of perfect information. This indicates that even with partial information many inheritance can be found to be incompatible in some situations. To our knowledge pedigrees larger than 30 bits have never been analyzed exact, [MDK01, ACCC02, Gud00].

JavaAllegro's on Perfect Information

The green graph in Figure 6-4 shows JavaAllegro's running times for the multi point probability computation for pedigrees with perfect information. In our test, the result of all individuals being genotyped was that only very few inheritance vectors were compatible with the given genotype information. The results shown in Figure 6-4 indicate, that JavaAllegro's running time in practice grows linear with the size of pedigrees with perfect information. Since JavaAllegro computes only the necessary contributions, and since perfect genotype information results in few compatible inheritance vectors - thus small MTBDDs - the linear growth it not surprising[16].

The table below contains the node counts for the MTBDDs representing multi point probabilities for the 5 largest pedigrees with perfect information. In the right-most column we have included the running time of multi point probability computation for the 5 markers.

[14] Of course we do not need a left contribution vector for the left-most marker, and similarly we do not need a right contribution vector for the right-most marker.

[15] See [CHI+02] and [Gud00] for more on founder couple reduction.

[16] Even though, we were surprised at first.

Pedigree size \ Marker #	0	1	2	3	4	Time
40 bits	1.237	4.859	80	733	1.733	25 s
41 bits	165	750	316	1.441	56	28 s
42 bits	534	1.155	183	251	546	30 s
43 bits	158	321	2517	179	156	28 s
44 bits	1.904	387	898	550	3.930	31 s

In human pedigrees perfect genotype information is unlikely to occur when considering just a few generations back, however, for lab studies of animals and insects the breeding process is controlled and all specimens are available for genotyping. For these types of studies, using MTBDDs for representation of probability distributions seems very applicable, since perfect information is available.

We have not performed multi point probability computation on pedigrees with perfect information larger than 44 bits, since we have not had access to such. However, the results indicate that we should expect only a linear growth in running time for even larger pedigrees, See Figure 6-4.

JavaAllegro's Performance on Partial Information

To generate partial information, we have used GenoType (see page 80), where we have set the probability of non-leaf individuals being genotyped to 50%.

Figure 6-5 on the next page depicts the results of generating random genotype information for an average[17] of 25 times for each pedigree in the range of 21 to 39 bits. The figure includes worst running times (blue graph), best running times (green graph), and average running time (red graph). The graph for the best running times follows the same pattern as for perfect genotype information. This indicates that only a very few inheritance vectors have been compatible with the given genotype information. This is a very interesting result, since in the randomly generated genotype information non-leaf individuals have only a 50 percent probability of being genotyped. This means that even for somewhat realistically genotyped human pedigrees using MTBDDs for representation and manipulation of probabilities seems efficient.

Even in the worst case, MTBDDs can provide an efficient representation of multi point probabilities. The highest running time resulted from a 31 bit pedigree for which computations took 2 hours and 13 minutes. Using the above extrapolation of Allegro's running this could be computed in 5 hours and 40 minutes provided that we had 120GB of primary memory. The resulting node counts of the MTBDDs in the 31 bit pedigree, for which multi point probabilities took of 2 hours and 13 minutes to compute, are summarized below:

Pedigree size \ Marker #	0	1	2	3	4	Time
31 bits	526.269	1.672	1.642	2.135	44.228	2h13m

From this it appears that marker 0 is responsible for the long running time due to the large MTBDD, and we naturally expect larger MTBDDs to result in longer running times. However, below we provide two test results for the 39 bit pedigree:

[17] For some of the pedigrees and some of the random genotype information, we were unable to perform single point analysis within a reasonable time frame.

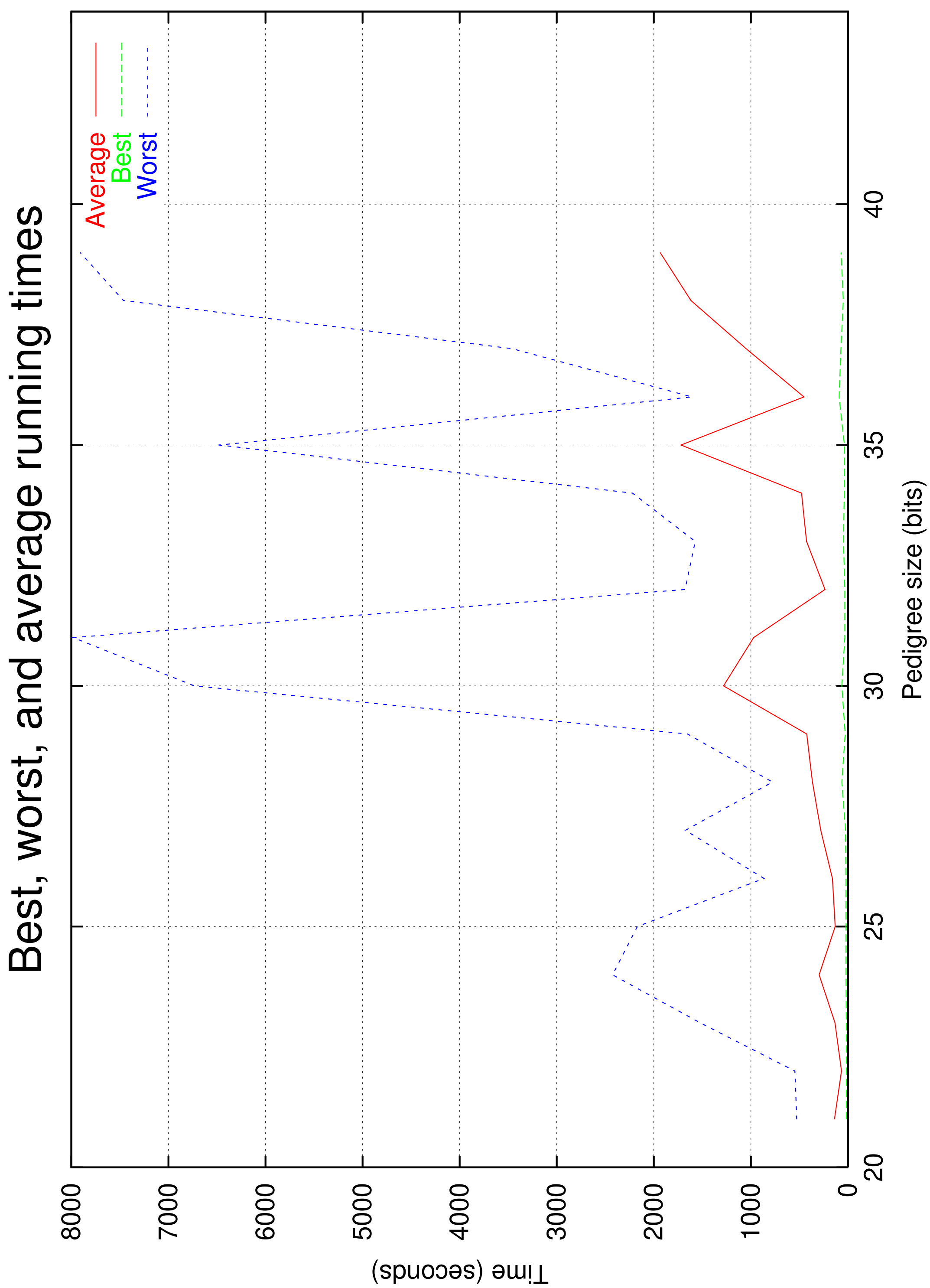

Figure 6-5: Experimental results for running JavaAllegro on pedigrees in the range of 21 bits to 39 bits.

Pedigree size \ Marker #	0	1	2	3	4	Time
39 bits test a	18.926	383	109.325	46.970	285.812	2h11m58s
39 bits test b	291.103	603	4.101	48.252	38.8261	13m35s

In this case there appears to be no direct relationship between the sizes of the MTBDDs and the running time of multi point probability computation. The reason behind these results could lay in the implementation. E.g. if the hash function which stores nodes in hash tables is not properly balanced, a considerable amount of time is spent looking for an free bucket in the hash table. We conclude that more experimentation is needed in order to give a more qualified explanation. However, using 13 minutes and 35 seconds to analyze a 39 bit pedigree with MTBDDs size up to 388.261 nodes is a significant improvement over current methods.

Before we could perform the multi point probability computation, we obviously had to perform single point analysis. It turned out that the single point probability computation became the bottle-neck for some pedigrees dependent on the genotype information. The single point algorithm implemented in JavaAllegro is FASTTREETRAVERSAL, [Gud00, CHI+02]. The running of this algorithm is dependent on the genotype information, but its worst case time complexity is exponential in the pedigree size. For large pedigrees some genotype information can result in only a small fraction, possibly none, of the inheritance vectors being incompatible. For such genotype information the single probability computation will a have near worst case running time, which practically makes the pedigree impossible to analyze. E.g. computing the single point probabilities of all inheritance vectors in a 38 bit pedigree would require approximately 31 days at a rate of 100.000 inheritance vectors per second[18]. We have not been able to generate partial genotype information for pedigrees larger than 39 bits, for which we were able to compute the single point probabilities within a reasonable time frame.

Our DD-based method does benefit from a large fraction of inheritance vectors having a probability of zero given the available genotype information. In the cases where the majority are all inheritance vectors are compatible with the given genotype information, our method might perform poorly. Hence, for a 38 bit pedigree, if all 2^{38} inheritance vectors have different probabilities, we need 2^{38} terminals in the MTBDD representing the probability distribution. Obviously, this is not feasible. However, in the case where not all inheritance vectors have different probabilities MTBDDs might still be able to represent probability distributions compactly compared to the explicit representation. To investigate this, however, we need a new, improved single point algorithm.

6.2.5 Summary

The major results of the experiments can be summarized as follows:

- Using a dual Pentium 1 GHz with 1GB of memory Allegro is only able to analyze pedigree up 25 bits. This is because time and memory requirements double for every additional bit in the input pedigree.

[18] 100.000 inheritance vectors per second is actually on average twice as many per second as we able to compute.

- For pedigrees with perfect genotype information the running time of JavaAllegro grows only linear in the size of the pedigree. A 44 bit pedigree has been analyzed in 31 seconds. As far as we know this is 12 bits larger than any other current method has been able to compute. Furthermore, as the results indicate, only a linear growth in practical running time is expected for pedigrees larger than 44 bit.
- For partial genotype information JavaAllegro has been able to analyze pedigrees up to 39 bits using less than 1GB of memory. For larger pedigrees the single point algorithm became the bottle-neck.

6.3 Summary

In this section we have presented two algorithms for performing multi point probability computation using MTBDDs, one for distributions over complete inheritance vectors, and one for distributions over founder reduced inheritance vectors. We have proved their correctness and shown how dynamic programming techniques can be applied in order to obtain a good performance. Our experimental results have shown that we have been able to analyze previously intractably large pedigrees both with perfect and partial genotype information. For pedigrees with perfect genotype information there appears to be a linear relationship between the practical running time and the pedigree size in bits. For pedigrees with partial genotype information, we have observed very diverse results. The running time for the multi point probability computation for pedigrees with partial genotype information has shown to be very sensitive to the number of compatible inheritance vectors.

Conclusion

Chapter 7

In this thesis we have given a short introduction to some the biological concepts behind linkage analysis, and we have given a summary of the formalization of linkage analysis developed in our previous work, [CHI$^+$02]. We have shown how multi point probability computation can be performed by the use of fast Fourier transform, the approach currently used in Allegro and GeneHunter.

We have developed a new method, namely the *recombination transform*, and it has been extended to be applicable to founder reduced inheritance vectors. This new method is faster than the former method based on Fourier transforms. The reason is the fast Fourier transform-based approach requires two transforms and one multiplication of the probability distribution to compute contributions, whereas the (founder reduced) recombination transform requires only one transform. We have proved the correctness of the recombination transform and shown how it can be rewritten by using grouped inheritance vectors in order to obtain an efficient algorithm.

Pedigree analysis software such as Allegro and GeneHunter use explicit array representation of probability distributions. One of the major limitations of this approach is the space complexity which is exponential in the size of the pedigree analyzed. To overcome this limitation, we have investigated the applicability a number of different DD-based data structures. The heuristics approach showed that decision diagram data structures, based on Shannon decomposition, can efficiently represent the sparseness found in some probability distributions, especially larger pedigrees, where a large fraction of inheritance vectors can be incompatible with the given genotype information. Our investigation also showed that the sizes of any of the data structures tested were insensitive to the variable orderings we evaluated.

Finally, we presented an MTBDD-based algorithm for computing contributions to single inheritance vectors, based on the recombination transform. We proved the correctness of the algorithm, and we modified it in order to make it applicable to probability distributions over founder reduced inheritance vectors. The algorithm for performing founder reduced recombination transform has also been proved correct. We showed how dynamic programming techniques can be used to speed-up computations, and if the variable ordering of inheritance vectors at every other marker is reversed, we can easily find inheritance vectors with non-zero probabilities in the optimal order.

We have implemented a multi point pedigree analyzer which uses MTBDD data structures for the representation of probability distributions, and we have implemented the MTBDD version of the recombination transform. Our experimental results have shown, that computing individual contributions is efficient for some pedigrees with some genotype information, namely genotype information which results in sparse probability distributions over inheritance vectors.

This method has enabled us to analyze pedigrees which were previously prohibitively large. For pedigrees with perfect information we have even observed a linear relationship between the pedigree size and the running time of multi point probability computation. In this case, we have analyzed the largest pedigree supplied by deCODE Genetics (44 bits) in 31 seconds. However, even larger pedigrees are expected to have a similarly low running time[1].

7.1 Future Work

Our work and results open for six major areas of further research:

An Implementation in a Compiled Language

JavaAllegro has served as an unoptimized prototype implementation to prove the applicability of Shannon-based DDs and the recombination transform. Since Java is an interpreted language its performance is nowhere near the performance of a compiled language like C. Even though our results indicate superior performance and open for analysis of priorly intractably large pedigrees, an implementation in a compiled language is most likely required in a commercial setting.

Further Analysis Based on Heuristics

In this thesis we have concentrated on the performance of the multi point probability computation part of linkage analysis. After this part a number of other types of analyses are applied in order to locate genes affecting a trait, such as haplotyping, likelihood inference, pair-wise IBD estimation, etc. Efficient algorithms for these kinds of analyzes in Shannon-based DDs are needed in order to complete the picture. The other types of analyzes have been not been within the scope of this report, however, it is likely that the other analyzes can benefit from MTBDD representation like multi point probability computation.

A New Single Point Algorithm

The fast tree traversal algorithm is used for computing single point probabilities in modern pedigree analysis packages, such as Allegro and Merlin, [Gud00, CHI$^+$02, ACCC02]. The algorithm operates in a top-down, breadth-first fashion with respect to the pedigree structure. It has proven efficient for pedigrees up to the size of 24 bits, [Gud00]. The running time of the algorithm, however, is dependent on both the pedigree and genotype data. In some cases incompatibilities might be rediscovered many times as the algorithm re-recurses down the pedigree. It has been our experience that for large pedigrees, the single point probability computation takes a significant part of the running time, sometimes many times more than multi point probability computation, depending on the genotype information. Hence, the current single point algorithm is the bottle-neck for some pedigrees and genotype information. We believe that a new algorithm is needed in order to be able to analyze large pedigrees efficiently.

[1] We modified the 44 bit pedigree with three extra children to obtain a 50 bit pedigree, which for perfect genotype information was analyzed in 32 seconds.

Recombination Transform with Founder Couple Reduction

Gudbjartsson showed in [Gud00] how inheritance vectors can not only be founder reduced but also *founder couple reduced*. This reduction only works for non-genotyped founder couples with grand-children in the pedigree analyzed, See [CHI+02]. Probability distributions and contribution vectors over founder couple reduced inheritance vectors might have more compact DD representation than non-reduced inheritance vectors. If this is the case, further research is necessary to determine whether the recombination transform can be extended to operate efficiently on both founder *and* founder couple reduced inheritance vectors.

Approximations

In [ACCC02] Abecasis et al. suggest a way to approximate multi point probability computations by assuming that no more than a single recombination is likely to occur between adjacent markers for dense genetic linkage maps, where the recombination fractions between markers are small[2]. This assumption might be valid for pedigrees below a certain size, however the limit for the number of recombinations, which can be considered likely, is of course not just a function of the recombination fraction but also of the pedigree size.

With a MTBDD implementation an alternative way of approximating the multi point probabilities can easily be implemented: Define a number, ϵ, which denotes the minimum distance between terminal values in the MTBDD. When probability/contribution, p, is inserted into the MTBDD a terminal is considered to represent p if it is labeled with a value in the range $[p - \epsilon \ ; \ p + \epsilon]$. With a large value for ϵ, depending on the values of the terminals, we can assume fewer terminals in an MTBDD, and hence more subgraph sharing. Since the time complexity of the recombination transform is linear in the number of nodes of the input, such an approximation is likely to improve the performance of the multi point probability computation. However, the performance and the statistical consequences for the further genetic analysis of such an approximation are subject for further research.

Distributed Linkage Analysis

Linkage analysis is relative easy to parallelize on a uniform memory access architecture. However, due to the size of the probability distributions and contributions vectors when using explicit array representations, it is hard to distribute on NUMA architectures and clusters using message passing, due to the size of interdependent data. E.g. since genotype information for every marker is used to compute the probability of inheritance patterns for every other marker, large amounts of data have to be transferred between nodes. The compact representation of probability distributions and contribution vectors, that Shannon-based DDs can offer, opens for the possibility of distributing multi point probability computations on NUMA architectures and message passing-based clusters, since the amount of data needed transferred is reduced.

[2] In [ACCC02] there is no indication of how small a recombination fraction has to be to be *small*.

LOD and NPL Score

APPENDIX A

A scoring function is a measure of the likelihood of linkage between a marker and a trait, for which the locus of the marker is known, but for which the locus or loci of the trait causing gene(s) are unknown. Note that when one is looking for the genetic cause of a trait a number of pedigrees are used.

It is not within the scope of this project to go into detail with the scoring functions, as they rely the statistical part of linkage analysis which is not our focus. We will, however, briefly describe two commonly used scoring functions: The *Log Likelihood of Odds* (LOD) score, and the *Non-Parametric Linkage* (NPL) score.

A.0.1 LOD Score

The LOD score is a measure of how plausible an observed set of data is given a model, [Ott99, page 38]. It is the logarithm of the odds of the observed data if linkage is assumed (recombination fraction $\theta < \frac{1}{2}$) compared to the *null-hypothesis* (no linkage) where the recombination fraction θ is equal to $\frac{1}{2}$. Formally the LOD score is, [Ott99, page 39]:

$$\mathrm{LOD}(\theta) = \log_{10}\Big(\frac{L(\theta)}{L(\frac{1}{2})}\Big), \tag{A-1}$$

where L is the likelihood of a hypothesis H given some observation data F, defined as: $L(H) = P(F|H)$. The odds in favor one hypothesis H_1 versus H_2 are expressed by the likelihood ratio:

$$R = \frac{L(H_1)}{L(H_2)}.$$

Maximum likelihood estimation is used to find a θ which maximises the LOD-score for all markers (See [Ott99, Chapter 3.3] for more on the statistics of LOD scores and maximum likelihood estimation). The probability of each inheritance vector $v \in \mathbf{v}$ is used in the expression for the score (See [Gud00] for a more detailed presentation of the LOD score computation).

A.0.2 NPL Score

The NPL score does not depend on a hypothesis (or a parameter) as the LOD score does. It depends on a *non-parametric scoring function* S such as S_{pair} or S_{all} defined by Kruglyak et. al in [KDRDL96]. Z_i is the standardized from S_i (the score for family i):

$$Z_i = \frac{S_i - \mu_i}{\sigma_i},$$

where μ_i is the mean of S_i, and σ_i is the standard deviation.

A formal definition of NPL is given by Gudbjartsson in [Gud00]:

$$\mathrm{NPL} = \frac{\sum_i \gamma_i \bar{Z}_i}{\sqrt{\sum_i \gamma_i^2}},$$

where $\sum_i$ is the sum over all families, γ_i is a family specific weight and $\bar{Z}_i$ is the expectation of Z_i conditioned on the observed genotype data. Hence:

$$\bar{Z}_i = \sum_{\mathbf{v_i}} P(\mathbf{v_i}|\mathcal{G}_i) Z_i(\mathbf{v_i}),$$

where $\mathbf{v_i}$ is the set of inheritance vectors for family i at a marker and $\mathcal{G}_i$ is the genotype information for the family at the marker.

The reader should note that the every element of the probability distribution $P(\mathbf{v}|\mathcal{G})$ is used in both scoring functions.

Definitions of DD Data Structures

Appendix B

In this appendix we provide an explanation and definition of all the data structures used in the analysis in Chapter 5. The different data structures are: MTBDDs, EVBDDs, BMDs, *BMDs, K*BMDs, and *HDDs.

We define a function, g, for which we present its representation according to each data structure. The exemplification is provided to give the reader an intuitive feel of the semantics of each data structure. g is defined over three boolean variables x, y, and z according to Table B-1.

x	y	z	$g(x,y,z)$
0	0	0	0
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	10
1	0	1	12
1	1	0	14
1	1	1	16

Table B-1: The common function for which each data structure is exemplified. Analytically g is defined as $g(x,y,z) = -2xyz + 10x + 2yz + 4y + 2z$.

Background

All DD data structures used in this analysis are, in one way or another, extensions on Binary Decision Diagrams (BDDs) as introduced by Bryant in [Bry86]. BDDs has a canonical representation of every function with signature $f : \mathbb{B}^n \to \mathbb{B}$. We now give a short introduction to BDDs.

Given the function $f(x_1, \ldots, x_n)$ over n boolean variables, the *cofactor* $f_{x_i\overline{x_j}x_k}$ is the function obtained by setting $x_i = 1, x_j = 0, x_k = 1$ and evaluating f for $(1 \leq i, j, k \leq n,\ i \neq j \neq k)$[1].

Example 39 (Cofactors) *Consider a function f over three boolean variables x_1, x_2, and x_3 defined as:*

[1] Notice that in Section 3.3.2 on page 30 we introduced a similar notion for vectors.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	*0*	*0*	*0*
1	*0*	*0*	*0*
0	*1*	*0*	*0*
1	*1*	*0*	*1*
0	*0*	*1*	*1*
1	*0*	*1*	*1*
0	*1*	*1*	*1*
1	*1*	*1*	*1*

f corresponds to the boolean function $x_1 \cdot x_2 + x_3$. The cofactor $f_{x_1\overline{x_3}}$ of f is the function:

x_2	$f_{x_1\overline{x_3}}(x_2)$
0	*0*
1	*1*

Using cofactors we can rewrite boolean functions with the *Shannon Decomposition* on any variable, x_i:

$$f = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}. \tag{B-1}$$

f_{x_i} and $f_{\overline{x_i}}$ are called the *positive* and *negative cofactors* of f, respectively. By performing the Shannon Decomposition recursively for all variables we obtain a full, binary decision tree with leaves $0, 1$ (logical false and true). Each internal node, v, represents a function, f^v. The left and right children of v represents the negative and positive cofactors of f^v, with respect to some variable x_i. This is called a *Shannon Tree* of f. Figure B-1 shows a Shannon Tree for the function in Example 39.

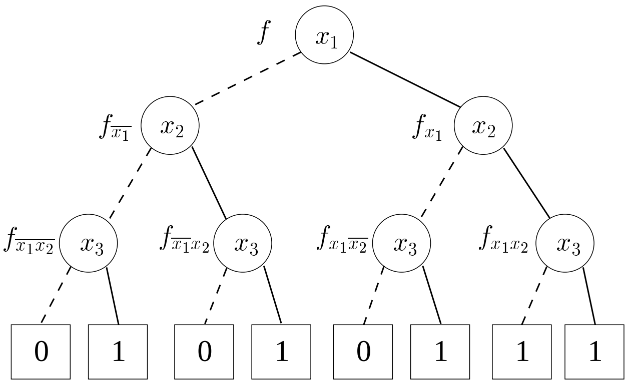

Figure B-1: A Shannon Tree for the function f in Example 39 representing the boolean function $x_1 \cdot x_2 + x_3$. Dashed lines indicates negative cofactor and solid lines indicate positive cofactor. The nodes are labeled with the variable which is expanded in each node. The function shown to the left of each node indicates the cofactor of f described by that node.

A Shannon Tree is said to be *ordered* if all nodes at a certain level expand with respect to the same variable. Note that the Shannon Tree in Figure B-1 is ordered. We obtain an *Ordered*

Binary Decision Diagram (OBDD) if we view the ordered Shannon Tree as a directed, acyclic graph directed from the root to the leaves. An OBDD is reduced (ROBDD) if all isomorphic subtrees at a given level are shared and redundant[2] nodes are removed. Figure B-2 shows a ROBDD representing the ordered Shannon Tree of Figure B-1. From now on we refer to ROBDDs simply as BDDs.

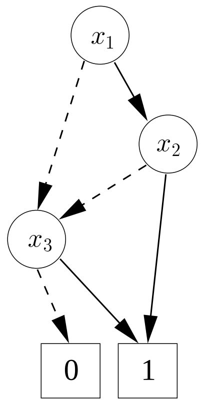

Figure B-2: An ROBDD representing the Shannon Tree of Figure B-1, that is, the boolean expression $x_1 \cdot x_2 + x_3$.

Each function with signature $f : \mathbb{B}^n \rightarrow \mathbb{B}$ has a canonical BDD representation, [Bry86]. Obviously, the BDD representation of a function over n variables has a worst-case size exponential in n. However, as mentioned earlier BDDs have proven very useful as a compact representation of functions over many variables.

Like BDDs, all data structures introduced in this section are represented as reduced binary trees. For this reason we give a definition of a DD data structure which serves as a generalization of all data structures introduced. This allows us to describe and reason about different data structures without explicitly defining them. Since only one data structure is used after the analysis, we have chosen to leave out formal definitions of all data structures in favor of giving the reader a more intuitive understanding.

Definition 22 (General DD Data Structure) *A general DD data structure over a set of variables VAR consists of a directed, acyclic graph G with a node set of terminals, T, with out-degree zero and non-terminals[3], V, with out-degree two.*

- *Each terminal node, n, is assigned a value in $\mathbb{Z}$ by the function $value : T \rightarrow \mathbb{Z}$.*
- *Each non-terminal, n, is a 5-tuple $\langle label(n), w_0, child_0(n), w_1, child_1(n)\rangle$ where:*
  - $label : V \rightarrow$ *VAR determines the variable assigned to VAR.*
  - $child_0 : V \rightarrow V \cup T$ *determines the 0-child of a node.*

[2] A node is redundant if $f_{x_i} = f_{\overline{x_i}}$.
[3] Sometimes referred to as inner nodes

- $w_0 \in \mathbb{Z}$ *is a weight assigned to edge leading to the 0-child.*
- $child_1 : V \to V \cup T$ *determines the 1-child of a node.*
- $w_1 \in \mathbb{Z}$ *is a weight assigned to edge leading to the 1-child.*

In Figure B-2 the BDD is ordered which is an important property to ensure cononicity. On the general data structure we define an ordering of the variables by an injection[4] $level$ from VAR to $\mathbb{N}$. A DD data structure is said to be *ordered* if:

$$\forall\, n \in V\ .\ level(label(n)) < level(label(child_i(n))), \tag{B-2}$$

where $i \in \{0, 1\}$. Since $label$ is not defined on terminals, we allow any node in a ordered DD data structure to have terminal children.

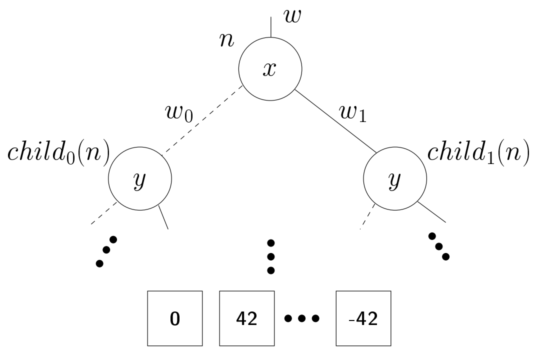

Figure B-3: A representation of a general DD data structure. Non-terminals are labeled by their corresponding variable and shaped as circles. Terminals are labeled by their value and shaped as squares. Dashed edges indicate 0-child relationship. Solid edges indicate 1-child relationship. Arrows are omitted as directions are always top-down.

Since all DD data structures are defined as directed, acyclic graphs, their semantics are most easily understood when the structures are represented graphically. We now give a brief introduction to our representation of such data structures. A general representation is depicted in Figure B-3. Consider a non-terminal node $n = \langle label(n), w_0, child_0(n), w_1, child_1(n)\rangle$, n is then labeled by $label(n)$, has a dashed edge labeled w_0 leading to $child_0(n)$ and a solid edge labeled w_1 leading to $child_1(n)$. This is the situation illustrated in Figure B-3. Terminal nodes are labeled by their value. All DD data structures are directed in a top-down fashion and in general we omit arrows on edges. If the parent of a node is unknown or irrelevant (as node n in Figure B-3) we denote the weight on the edge leading to this node simply by w. Non-terminals are shaped as circles and terminals are shaped as squares.

Some data structures in the following analysis do not allow weight on edges and in this case we just ignore them, which is the case for BDDs.

[4] A function f is an injection if for every elements a_1, a_2 in the domain of f $f(a_1) = f(a_2)$ implies $a_1 = a_2$. Such a function is also called one-to-one.

In the introduction of each DD data structure we show its representation of the function defined in Table B-1. Since all data structures are instances of the general DD data structure of Definition 22 the difference appears in the interpretation of nodes as functions. In the following we focus on explaining the function that a node in each data structure represents.

For this purpose we define a function f on each data structure which determines the function of a node with an edge with weight w leading to it. That is, for each data structure we define a recursive function:

$$f : \mathbb{Z} \times V \cup T \to \mathbb{Z}.$$

For data structures which do not allow edge weights we leave out $\mathbb{Z}$ from the domain of the function, that is, f gets signature $f : V \cup T \to \mathbb{Z}$.

For data structure which have two integers assigned to each edge we define f as $f : \mathbb{Z}^2 \times V \cup T \to \mathbb{Z}$.

Example 40 (Function of BDD) *We saw earlier how the function determined by a BDD coincides with the Shannon Decomposition of the function. For this reason, the function f on any node, n, of a BDD is defined as:*

$$f(\langle n \rangle) = \begin{cases} value(n) : \ n \in T \\ (1-x) \cdot f(\langle child_0 \rangle) + x \cdot f(\langle child_1(n) \rangle) : \ n \in V \ \text{and} \ label(n) = x. \end{cases} \quad \text{(B-3)}$$

B.1 MTBDD

The idea of extending BDDs to allow non-binary terminals was introduced in 1993 by Fujita et al. in [FYC+93] to perform Walsh transformations on BDDs. This article allowed terminals to represent the set of integers whereas another article published the same year, [FYM93], allowed the terminals to be any finite subset, $\mathbb{R}'$, of the reals, $\mathbb{R}$. We will just focus on functions onto $\mathbb{Z}$.

In general MTBDDs can be seen as the obvious generalization of BDDs, where terminals can take on values other than 0 and 1. Thus MTBDDs are just the Shannon Decomposition of functions with signature $f : \mathbb{B}^n \to \mathbb{Z}$ and we define the function of any node n with $label(n) = x$ in a MTBDD as:

$$f(\langle n \rangle) = \begin{cases} value(n) : \ n \in T \\ (1-x) \cdot f(\langle child_0 \rangle) + x \cdot f(\langle child_1(n) \rangle) : \ n \in V \ \text{and} \ label(n) = x. \end{cases} \quad \text{(B-4)}$$

MTBDDs share the canonicity of BDDs, that is, each function with signature $f : \mathbb{B}^n \to \mathbb{Z}$ has a unique MTBDD representation.

Example 41 (Function Representation with MTBDD) *The MTBDD representation of the function $g(x, y, z) = -2xyz + 10x + 2yz + 4y + 2z$ from Table B-1 on page 117 is given in*

Figure B-4. According to Equation B-4 the value of g(1,1,0) is given as:

$$\begin{aligned} f(0,0,1) &= 0 \cdot f(\langle child_0(n_1)\rangle) + 1 \cdot f(\langle child_1(n_1)\rangle) \\ &= 0 \cdot f(\langle child_0(n_3)\rangle) + 1 \cdot f(\langle child_1(n_3)\rangle) \\ &= 1 \cdot f(\langle child_0(n_7)\rangle) + 0 \cdot f(\langle child_1(n_7)\rangle) = 14 \end{aligned}$$

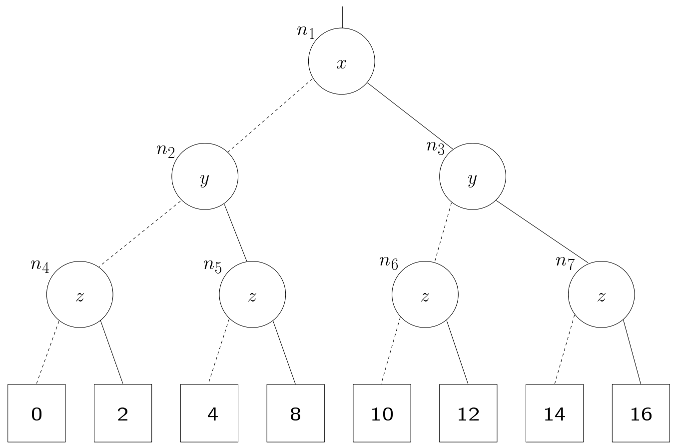

Figure B-4: MTBDD representing the function $g(x, y, z) = -2xyz + 10x + 2yz + 4y + 2z$ of Table B-1 on page 117.

B.2 EVBDD

EVBDDs were introduced by Lai et. al. in [LPV93] for symbolic representation of arithmetic functions over binary variables. Like MTBDDs, EVBDDs provide a canonical representation[5] for any function with signature $f : \mathbb{B}^n \to \mathbb{Z}$. Later, in 1994 Lai et al. published an article, [LPV94], providing application domains for EVBDDs such as integer linear programming, function decomposition, and spectral transformation. The latter of the three seems applicable in within linkage analysis and is our main reason for the inclusion of this data structure in the analysis.

Contrary to MTBDDs, EVBDDs allow only one terminal node with value 0. Instead positive edges can hold any value in $\mathbb{Z}$ and negative edges only hold zero. To find the function value of a given assignment of binary values to the variables in VAR, EVBDDs are traversed similarly to MTBDDs, but adding the encountered edge weights. Edge values with this interpretation are called *additive* edge weights. More precisely, the function described by any node n, with

[5] Canonicity requires the EVBDD to be reduced and ordered.

incoming weight w in an EVBDD is defined recursively as:

$$f(\langle w,n\rangle) = \begin{cases} w: \ n \in T \\ w + (1-x)\cdot f(\langle 0, child_0\rangle) + x \cdot f(\langle w_1, child_1(n)\rangle) : n \in V \text{ and } label(n) = x. \end{cases} \tag{B-5}$$

[LPV94] describes an EVBDD in a special *flattened* version, which is simply an EVBDD without values on edges since the values are moved down into the leaf allowing for multiple terminals(?!). From this it follows that the flattened EVBDD is *identical* to the earlier introduced (1993) MTBDD.

Example 42 (Function Representation with EVBDD) *This example illustrates the EVBDD representation of the function $g(x,y,z) = -2xyz + 10x + 2yz + 4y + 2z$ given in Table B-1 on page 117. The EVBDD graph of f is depicted in Figure B-5. According to Equation B-5 the function value of $g(1,1,0)$ is given as:*

$$\begin{aligned} g(1,1,0) &= 0 + 0 \cdot f(\langle 0, child_0(n_1)\rangle) + 1 \cdot f(\langle 10, child_1(n_1)\rangle) \\ &= 10 + 0 \cdot f(\langle 0, child_0(n_3)\rangle) + 1 \cdot f(\langle 4, child_1(n_3)\rangle) \\ &= 10 + 4 + 1 \cdot f(\langle 0, child_0(n_5)\rangle) + 0 \cdot f(\langle 2, child_1(n_5)\rangle) \\ &= 10 + 4 + 0 + 0. \end{aligned}$$

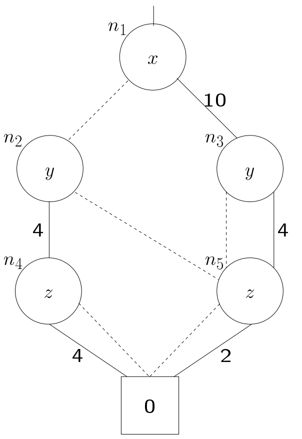

Figure B-5: EVBDD representing the function $g(x,y,z) = -2xyz+10x+2yz+4y+2z$ of Table B-1 on page 117.

B.3 BMD

BMDs were introduced in 1994 by Bryant et. al., [BC94], to represent functions with signature $f : \mathbb{B}^n \to \mathbb{Z}$. It was developed specifically for verification of arithmetic circuits such as multipliers. BMDs have proven more compact for the representation of such circuits than other DD approaches, [BC94].

The latter has been the main motive for including BMDs in our analysis, since we know that the probability vector for multi point probability computation can have many different entries. Thus BMDs may be more compact than both MTBDDs and EVBDDs.

In structure, BMDs are identical to MTBDDs, but the semantics of nodes is somewhat different.

Any node, n, with $label(n) = x$, in a BMD determines the function:

$$f(\langle n\rangle) = \begin{cases} value(n) : n \in T \\ f(\langle child_0(n)\rangle) + x \cdot f(\langle child_1(n)\rangle) : n \in V \text{ and } label(n) = x. \end{cases} \tag{B-6}$$

Let n, with $label(n) = x \in$ VAR, be the root node of a BMD describing a function f, then $child_0(n)$ describes the negative cofactor of f with respect to x, that is $f_{\overline{x}}$, since only $child_0(n)$ is evaluated if x is set to 0. For n to describe f, it then follows that $child_1(n)$ describes the subtraction between the positive and the negative cofactor of f with respect to x, that is $f(\langle child_1(n)\rangle) = f_x - f_{\overline{x}}$. Let us clarify this with and example. Let ψ be an assignment of boolean values to the variables in VAR, where $\psi(x) = 1$. Evaluating $f(\langle n\rangle)$ from (B-6) we arrive at:

$$f(\langle n\rangle) = f(\langle child_0(n)\rangle) + 1 \cdot f(\langle child_1(n)\rangle) = f_{\overline{x}} + f_x - f_{\overline{x}} = f_x.$$

If $\psi(x) = 0$ we get:

$$f(\langle n\rangle) = f(\langle child_0(n)\rangle) + 0 \cdot f(\langle child_1(n)\rangle) = f_{\overline{x}}.$$

For any function f, the decomposition of f with respect to x defined as follows:

$$f = f_{\overline{x}} + x \cdot (f_x - f_{\overline{x}}), \tag{B-7}$$

is known as the *moment decomposition* of f, where $f_{\overline{x}}$ is the *constant* moment and $f_x - f_{\overline{x}}$ is the *linear* moment, [BC94]. BMDs have a canonical representation of every function in terms of its moment decomposition.

The moment decomposition of Equation B-7 actually follows from the Shannon Decomposition by evaluation the expression:

$$\begin{aligned} f &= (1-x) \cdot f_{\overline{x}} + x \cdot f_x = f_{\overline{x}} - x \cdot f_{\overline{x}} + x \cdot f_x \\ &= f_{\overline{x}} + x \cdot (f_x - f_{\overline{x}}). \end{aligned}$$

Example 43 (Function Representation with BMD) *The moment decomposition of the function $g(x, y, z) = -2xyz + 10x + 2yz + 4y + 2z$ from Table B-1 on page 117 is given by the BMD representation in Figure B-6. According to Equation B-6 the value of $g(1, 1, 0)$ is given* *as:*

$$
\begin{aligned}
g(1,1,0) &= f(\langle child_0(n_1)\rangle) + 1 \cdot f(\langle child_1(n_1)\rangle) \\
&= (f(\langle child_0(n_2)\rangle) + 1 \cdot f(\langle child_1(n_2)\rangle)) + f(\langle child_1(n_1)\rangle) \\
&= ((f(\langle child_0(n_5)\rangle) + 0 \cdot f(\langle child_1(n_5)\rangle)) + f(\langle child_1(n_2)\rangle)) + f(\langle child_1(n_1)\rangle) \\
&= (0 + 0 \cdot 2 + f(\langle child_1(n_2)\rangle)) + f(\langle child_1(n_1)\rangle) \\
&= (f(\langle child_0(n_4)\rangle) + 0 \cdot f(\langle child_1(n_4)\rangle)) + f(\langle child_1(n_1)\rangle) \\
&= (4 + 0 \cdot 2) + f(\langle child_1(n_1)\rangle) \\
&= 4 + (f(\langle child_0(n_3)\rangle) + 1 \cdot f(\langle child_1(n_3)\rangle)) \\
&= 4 + (10 + f(\langle child_1(n_3)\rangle)) \\
&= 14 + (f(\langle child_0(n_6)\rangle) + 0 \cdot f(\langle child_1(n_6)\rangle)) = 14 + (0 + 0 \cdot (-2)) = 14.
\end{aligned}
$$

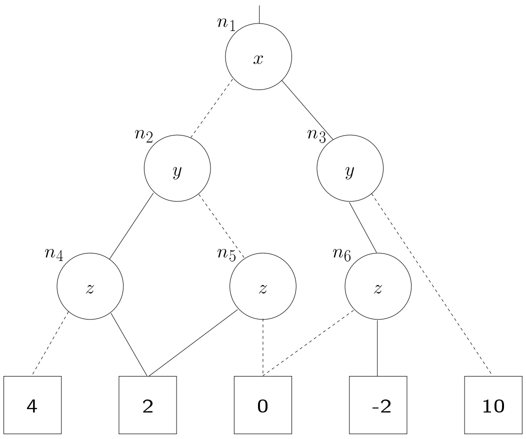

Figure B-6: BMD representing the function $g(x, y, z) = -2xyz + 10x + 2yz + 4y + 2z$ of Table B-1 on page 117.

B.4 *BMD

In the same paper which proposed BMDs, [BC94], Bryant et. al. proposed an extension to BMDs allowing multiplicative weights on edges. *BMDs have proven successful in compactly representing functions which have exponential BMD representations. These are in particular word-level functions, that is, the function value depends on the integer interpretation of boolean variables. E.g. the function 2^X, where X is the integer interpretation of e.g. three boolean variables. *BMDs build on the idea of EVBDDs to introduce weights on edges, but contrary to EVBDDs the edge weights are multiplicative, that is, the function represented by a node is multiplied by the weight on the edge leading to it. Formally, any node n with incoming edge weight w in a *BMD represents the function:

$$f(\langle w, n\rangle) = \begin{cases} w \cdot value(n) : n \in T \\ w \cdot (f(\langle w_0, child_0(n)\rangle) + x \cdot f(\langle w_1, child_1(n)\rangle)) : n \in V \text{ and } label(n) = x. \end{cases} \tag{B-8}$$

Introducing weights on edges complicates the set of rules assuring canonicity of the data structure[6]. Imagine $w = 1$ and w_0 and w_1 having c as their greatest common divisor, then we could describe the same function by $w' = c$, $w'_0 = \frac{w_0}{c}$, and $w'_1 = \frac{w_1}{c}$. The latter set of weights are known as *normalized weights*. To ensure canonicity all weights have to be normalized.

Example 44 (Function Representation with *BMD) *The linear function $g(x, y, z) = -2xyz + 10x + 2yz + 4y + 2z$ of Table B-1 on page 117 represented as a *BMD is depicted in Figure B-7. Note that the *BMD provides a more compact representation than the BMD. According to Equation B-8 we can retrieve the function value of $g(1, 1, 0)$:*

$$\begin{aligned} g(1,1,0) &= f(\langle 2, n_1\rangle) = 2 \cdot (f(\langle 1, child_0(n_1)\rangle) + 1 \cdot f(\langle 1, child_1(n_1)\rangle)) \\ &= 2 \cdot (f(\langle 1, n_2\rangle) + 1 \cdot (f(\langle 1, child_0(n_3)\rangle) + 1 \cdot f(\langle -1, child_1(n_3)\rangle))) \\ &= 2 \cdot (f(\langle 1, n_2\rangle) + (5 - f(\langle 1, child_0(n_5)\rangle))) \\ &= 2 \cdot (f(\langle 1, n_2\rangle) + (5 - 0)) \\ &= 2 \cdot ((f(\langle 1, n_5\rangle) + 1 \cdot f(\langle 1, n_4\rangle)) + 5) \\ &= 2 \cdot (0 + 2 + 5) = 2 \cdot 7 = 14 \end{aligned}$$

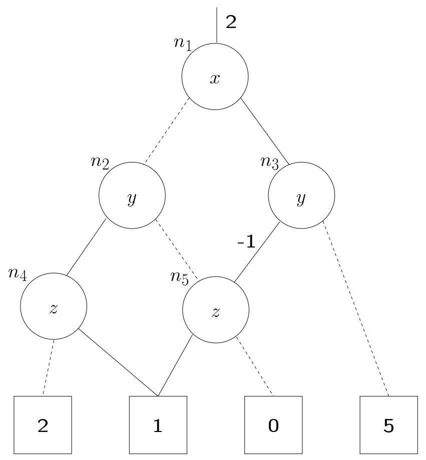

Figure B-7: *BMD representing the function $g(x, y, z) = -2xyz + 10x + 2yz + 4y + 2z$ of Table B-1 on page 117.

[6] One such set of rules are defined in [BC94].

TUDD supports a special version of *BMDs which only allows edge weight to be positive. This variant is called p*BMD. We have chosen not to include this data structure in the analysis.

B.5 K*BMD

In 1995 Drechsler introduced a generalization of *BMDs called Kronecker *BMDs, [RBS95]. The proposed data structure makes use additive as well as multiplicative edge weights. Obviously, this allows for greater sharing of common subgraphs for some domains. Furthermore, K*BMDs support different decomposition types for every variable, thus giving more flexibility. This makes K*BMDs applicable for both representations of bit and word level functions, [RBS95]. The three types of decompositions are *Shannon Decomposition, positive Davio decomposition*, and *negative Davio decomposition*. Positive Davio decomposition corresponds to the moment decomposition (B-7 on page 124) and negative Davio decomposition similar to positive except that the 1-child is only expanded when the variable is 0, that is:

$$f = f_x + (1-x) \cdot (f_{\overline{x}} - f_x). \tag{B-9}$$

Note that for both positive and negative Davio decomposition the 0-child is always included and the 1-child is included depending on the variable assignment. This implies that the 0-child holds the positive cofactor for negative Davio decomposition. Since this confuses the intuitive understanding of 0-child and 1-child these are sometimes referred to as *low* and *high* children.

Like EVBDDs, K*BMDs only allow one terminal, the 0-terminal.

The function, f, determined by any node, n, of a K*BMD depends on the decomposition rule for the variable corresponding to n. For any node, n, with incoming additive and multiplicative weights (w^a, w^m)[7] the function $f(\langle (w^a, w^m), n\rangle)$ is determined by:

- if n is terminal:

$$w^a \tag{B-10}$$

- if n is non-terminal with $label(n) = x$ and expanded with the Shannon Decomposition:

$$w^a + w^m \cdot ((1-x) \cdot f(\langle (0,1), child_0(n)\rangle) + x \cdot f(\langle (w_1^a, w_1^m), child_1(n)\rangle)). \tag{B-11}$$

- if n is non-terminal with $label(n) = x$ and expanded with the positive Davio decomposition:

$$w^a + w^m \cdot (f(\langle (0,1), child_0(n)\rangle) + x \cdot f(\langle (w_1^a, w_1^m), child_1(n)\rangle)). \tag{B-12}$$

- if n is non-terminal with $label(n) = x$ and expanded with the negative Davio decomposition:

$$w^a + w^m \cdot (f(\langle (0,1), child_0(n)\rangle) + (1-x) \cdot f(\langle (w_1^a, w_1^m), child_1(n)\rangle)). \tag{B-13}$$

[7] On additive weights we super scribe a and for multiplicative weights we super scribe m.

Notice that edges to 0-child always have additive weight 0 and multiplicative weight 1. This is a requirement to ensure canonicity of the representation. For a complete set of canonicity rules we refer to [RBS95].

The different decomposition rules have the consequence that if the Shannon Decomposition is chosen for every variable we arrive at EVBDDs with the addition of multiplicative edge weights. This way we can guarantee representations which are never larger than EVBDDs, since setting all multiplicative weights to 1 corresponds exactly to EVBDDs. On the other hand, if we choose only positive Davio decomposition we arrive at *BMDs with the addition additive edge weights. For the same reason as for Shannon Decomposition using only positive Davio decomposition we can represent functions which graphs that are never larger than *BMDs.

This does not mean that EVBDDs and *BMDs should be excluded from the analysis, since K*BMDs require more space per node than both EVBDDs and *BMDs. Thus, if the number of nodes in a K*BMD is basically the same as for the corresponding EVBDD, the EVBDD is preferable to the K*BMD.

Example 45 (Function Representation with K*BMD) *This example illustrates the a K*BMD representation of the function $g(x, y, z) = -2xyz + 10x + 2yz + 4y + 2z$ from Table B-1 on page 117. We have chosen only to illustrate the function for positive Davio decomposition such that the graph can be compared to Figure B-7. g's representation as a K*BMD is depicted in Figure B-8. According to Equation B-12 the function value of $g(1,1,0)$ is given as:*

$$\begin{aligned}
g(1,1,0) &= f(\langle(0,2),n_1\rangle) = 0 + 2\cdot(f(\langle(0,1),child_0(n_1)\rangle) + 1\cdot f(\langle(5,-1),child_1(n_1)\rangle))\\
&= 2\cdot(f(\langle(0,1),n_2\rangle) + 5 - 1\cdot(f(\langle(0,1),child_0(n_3)\rangle) + 1\cdot f(\langle(0,1),child_1(n_3)\rangle)))\\
&= 2\cdot(f(\langle(0,1),n_2\rangle) + 5 - (0 + f(\langle(0,1),child_0(n_4)\rangle)))\\
&= 2\cdot(f(\langle(0,1),n_2\rangle) + 5 - (0+0))\\
&= 2\cdot((f(\langle(0,1),child_0(n_2)\rangle) + 1\cdot f(\langle(2,1),child_1(n_2)\rangle)) + 5)\\
&= 2\cdot((f(\langle(0,1),child_0(n_4)\rangle) + 2 + 1\cdot(f(\langle(0,1),child_0(n_4)\rangle)) + 5)\\
&= 2\cdot(0+2+0+5) = 2\cdot 7 = 14
\end{aligned}$$

We have chosen to evaluate K*BMDs in three turns, one for each decomposition rule. This is because there is no obvious, qualified way of choosing different decomposition rules for the different variables (or bits) in inheritance vectors.

TUDD provides a special variant of K*BMDs, called pK*BMD, which only allows positive multiplicative weights. We have chosen to include this data structure in the analysis.

B.6 *HDD

In 1995 Clarke et al. introduced *Hybrid Decision Diagrams* (HDDs). HDDs are in nature very similar to K*BMDs in the sense that they both allow Shannon, positive Davio, and negative

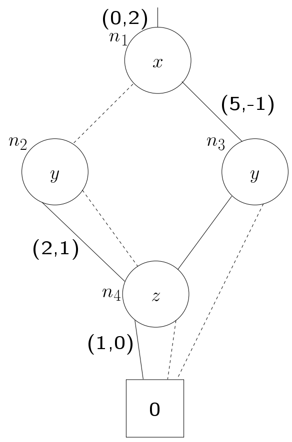

Figure B-8: K*BMD representing the function $g(x, y, z) = -2xyz+10x+2yz+4y+2z$ of Table B-1 on page 117. For any weight, (a, m), a is the additive weight and m is the multiplicative weight. Edges without weights are always $(0, 1)$.

Davio decompositions in different nodes in the graph. The only two differences are, that HDDs allow neither additive or multiplicative weights on edges, and HDDs allow other terminals than the 0-terminal. From this is follows that a HDD with the Shannon decomposition in every node is simply a MTBDD, and a HDD with positive Davio decomposition in every node is simply a BMD. We choose instead of regular HDDs to evaluate *HDDs, which are simply HDDs that allow multiplicative edge weights. This way we can construct the last variant of MTBDDs, namely MTBDDs with multiplicative edges weights.

The function, f, determined by any node, n, of a *HDD depends on the decomposition rule for the variable corresponding to n. For any node, n, with incoming multiplicative weight w the function $f(\langle w, n\rangle)$ is determined by:

- if n is terminal:
$$w \cdot value(n) \tag{B-14}$$
- if n is non-terminal with $label(n) = x$ and expanded with the Shannon Decomposition:
$$w \cdot ((1 - x) \cdot f(\langle 1, child_0(n)\rangle) + x \cdot f(\langle w_1, child_1(n)\rangle)). \tag{B-15}$$
- if n is non-terminal with $label(n) = x$ and expanded with the positive Davio decomposition:
$$w \cdot (f(\langle 1, child_0(n)\rangle) + x \cdot f(\langle w_1, child_1(n)\rangle)). \tag{B-16}$$
- if n is non-terminal with $label(n) = x$ and expanded with the negative Davio decomposition:
$$w \cdot (f(\langle 1, child_0(n)\rangle) + (1 - x) \cdot f(\langle w_1, child_1(n)\rangle)). \tag{B-17}$$

Notice, that negative children have only weight 1. This is necessary to ensure the canonicity of the data structure.

Example 46 (Function Representation with *HDD) *This example illustrates the a *HDD representation of the function $g(x,y,z) = -2xyz + 10x + 2yz + 4y + 2z$ from Table B-1 on page 117. We have chosen only to illustrate the function for Shannon decomposition such that the graph can be compared to Figures B-4 and B-5 to investigate the difference between the three types of MTBDDs. g's representation as a *HDD is depicted in Figure B-9. According to Equation B-15 the function value of $g(1,1,0)$ is given as:*

$$\begin{aligned} g(1,1,0) &= f(\langle 2, n_1\rangle) = 2 \cdot (1 \cdot f(\langle 1, child_1(n_1)\rangle)) \\ &= 2 \cdot (1 \cdot (1 \cdot f(\langle (0,1), child_1(n_3)\rangle))) = 2 \cdot (1 \cdot (1 \cdot (f(\langle 1, n_7\rangle))) \\ &= 2 \cdot (1 \cdot (1 \cdot (1 \cdot 7 + 0 \cdot 8))) = 14 \end{aligned}$$

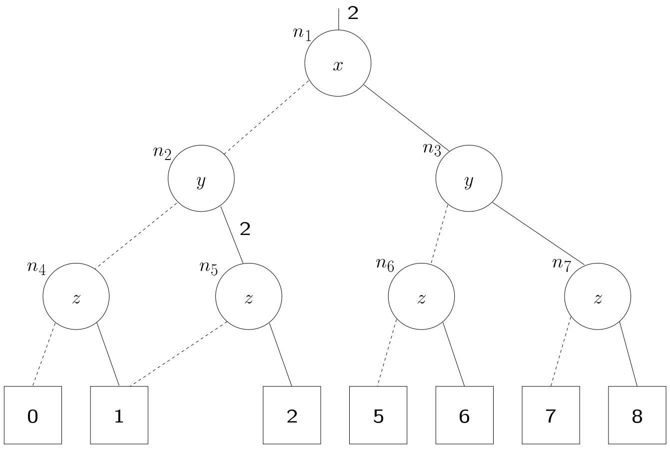

Figure B-9: *HDD representing the function $g(x,y,z) = -2xyz + 10x + 2yz + 4y + 2z$ of Table B-1 on page 117. Any weight, m, is a multiplicative weight. Edges without a weight are always 1.

Bibliography

[ACCC02] Goncalo R. Abecasis, Stacey S. Cherny, William O. Cookson, and Lon R. Cardon. Merlin - rapid analysis of dense genetic maps using sparse gene flow trees. *Nature Genetics Vol. 30*, pages 97–101, January 2002.

[All91] R. B. J. T. Allenby. *Rings, Fields and Groups - An Introduction to Abstract Algebra*. Edward Arnold, 2nd edition, 1991.

[BC94] R. E. Bryant and Y. A. Chen. Verification of arithmetic circuits with binary moment diagrams. Technical Report CMU-CS-94-160, Carnegie Mellon University, 1994.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, pages 142–170, 1992.

[BO98] Andreas D. Baxevanis and B.F. Francis Ouellette. *Bioinformatics - A practical guide to the analysis of genes and proteins*. Wiley-Interscience, 1998.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, pages 677–691, 1986.

[Cai71] Thomas W. Cairns. On the fast fourier transform on finite abelian groups. *IEEE Transaction on Computers*, 20:569–571, 1971.

[CB] Yirng-An Chen and Randal E. Bryant. *pbhd: An efficient graph representation for floating point circuit verification.

[CFZ95a] E. M. Clarke, M. Fujita, and X. Zhao. Applications of multi-terminal binary decision diagrams, 1995.

[CFZ95b] E.M. Clarke, M. Fujita, , and X. Zhao. Hybrid decision diagrams - overcoming the limitations of mtbdds and bmds. *In Int'l Conf. on CAD*, pages 159–163, 1995.

[CHI+02] Anders Lyhne Christensen, Jens 'monsteret fra Lundbykrat' Hansen, Anna Ingólfsdóttir, Jacob Johnsen, John Knudsen, and Jacob Illum Rasmussen. A formalization of linkage analysis. Technical Report RS-02-7, BRICS, February 2002. vi+109 pp.

[Cur01] Dave Curtis. Fundamental concepts in genetics, http://www.mds.qmw.ac.uk/statgen/dcurtis/lectures/introgen.html, October 2001.

[dNC01] deCode News Center. http://www.decode.com/news/releases/, November 2001.

[Fra94] John B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley, 5th edition, 1994.

[FYC+93] Masahiro Fujita, Jerry Chih-Yuan Yang, Edmund M. Clarke, Xudong Zhao, and Patrick McGeer. Fast spectrum computation for logic functions using binary decision diagrams. *Proc. IFIP WG 10.5 Workshop Applications of the Reed-Muller Expansion in Circuit Design*, pages 82–85, 1993.

[FYM93] Masahiro Fujita, Jerry Chih-Yuan Yang, and Patrick McGeer. Multi terminal binary decision diagrams: An efficient data structure for matrix representation. *Proc. IFIP WG 10.5 Workshop Applications of the Reed-Muller Ecpansion in Circuit Design*, pages 82–85, 1993.

[Gud00] Daniel Fanner Gudbjartsson. Multipoint Linkage Analysis based on Allele Sharing Models, 2000.

[GV01] Jaco Geldenhurs and Antti Valmari. Techniques for smaller intermediary bdds. *Concur 2001 - Concurrency Theory, 12th Internation Conference*, pages 233–247, August 2001.

[Hör97] Stefan Höreth. TUDD package: http://www.rs.e-technik.th-darmstadt.de/~sth/download.html, 1997.

[How01] Kenneth B. Howell. *Principles of Fourier Analysis*. Chapman & Hall/CRC, 2001.

[JH96] Thomas Jakobsen and Carlo Harpes. Bounds on non-uniformity measures for generalized linear cryptanalysis and partitioning cryptanalysis. *Pragocrypt '96 Proceedings*, pages 467–479, 1996.

[JIS93] Robert W. Cottingham Jr., Ramana M. Idury, and Alejandro A. Schaffer. Faster sequential genetic linkage computations. *American journal of human genetics*, 53:252–263, 1993.

[Joz98] R Jozsa. Quantum algorithms and the fourier transform. *Proc Roy Soc Lond A*, pages 323–337, January 1998.

[JSD01] Dragan Jankovic, Radomir S. Stankovic, and Rolf Drechsler. Decision diagram method for calculation of pruned walsh transform. *IEEE Transactions on computers*, 50(2):147–157, February 2001.

[KC97] William S. Klug and Michael R. Cummings. *Concepts of Genetics 5th edition*. Prentice Hall, 1997.

[KDRDL96] Leonid Kruglyak, Mark J. Daly, Mary Pat Reeve-Daly, and Eric S. Lander. Parametric and Nonparametric Linkage Analysis: A Unified Multipoint Approach. *American Journal of Human Genetics*, 58:1347–1363, 1996.

[KKNP01] Joost-Peiter Katoen, Marta Kwaitkowska, Gethin Norman, and David Parker. Faster and symbolic CTMC model checking. page 18, 2001.

[KL98] Leonid Kruglyak and Eric S. Lander. Faster multipoint linkage analysis using fourier transforms. *Journal of Computational Biology*, 5(1):1–7, 1998.

[KNPS99] Marta Z. Kwiatkowska, Gethin Norman, David A. Parker, and Roberto Segala. Symbolic model checking of concurrent probabilistic systems using MTBDDs and simplex. Technical Report CSR-99-1, University of Birmingham, 1999.

[Lan97] Kenneth Lange. *Mathematical and Statistical Methods for Genetic Analysis*. Springer, 1997.

[Lay97] David C. Lay. *Linear Algebra and its Applications*. Addison Wesley Longman, Inc., second edition, 1997.

[LG87] Eric S. Lander and Philip Green. Construction of Multilocus Genetic Linkage Maps in Humans. *Proc. Natl. Acad. Sci.*, 84:2363–2367, 1987.

[LPV93] Y-T Lai, M. Pedram, and S. Vrudhula. Edge-valued binary decision diagrams: Theory and applications. Technical report, University of Southern California, August 1993.

[LPV94] Y. Lai, M. Pedram, and S. Vrudhula. Evbdd-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Transactions on CAD*, 13(8):959–975, 1994.

[LWPY99] Kim G. Larsen, Carsten Weise, Justin Pearson, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6:271–298, 1999.

[MBS95] C. Meinel, J. Bern, and A. Slobodova. Efficient obdd-based boolean manipulation in cad beyond current limits, 1995.

[MDK01] Kyriacos Markianos, Mark J. Daly, and Leonid Kruglyak. Efficient multipoint linkage analysis through reduction of inheritance space. *American journal of human genetics*, 68:963–977, 2001.

[Mor94] Norman Morrison. *Introduction to Fourier Analysis*. Wiley-Interscience, 1994.

[Ott99] Jurg Ott. *Analysis of Human Genetic Linkage,Third Edition*. The Johns Hopkins University Press, 1999.

[PSAT92] William H. Press and Brain P. Flannery Saul A. Teukolsky, William T. Vetterling. *Numerical Recipes in FORTRAN 77: The Art of Scientific Computing 2nd ed.* Cambridge University Press, 1992.

[Rad] Mark Karpovsky Radomir. Recent advent in spectral techniques in binary and multiple-valued switching theory.

[RBS95] R. Drechsler, B. Becker, and S. Ruppertz. K*BMDs: a new data structure for verification. In *IFI WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Frankfurt, Germany, 1995.

[RRK01] Jacob I. Rasmussen, Joakim Recht, and John Knudsen. Multi-attributed interval decision diagrams. *http://www.cs.auc.dk/∼illum/maidd/index.html*, May 2001.

[SK97] Radomir S. Stankovic and Bogdan J. Kalkowski. Fft and decision diagrams based methods for calculation of spectral transforms. *ICICS, Singapore 9-12 September 1997*, pages 241–245, 1997.

[SR99] Tom Strachan and Andrew P. Read. *Human Molecular Genetics 2*. Wiley-Liss, 1999.

[TP96] P. Tafertshofer and M. Pedram. Factored edge-valued binary decision diagrams, 1996.

List of Figures

List of Tables

Glossary

Allele: Term used for the (maybe unknown) gene that is inherited from the father and the mother respectively. A single allele for each gene locus (allele) is inherited separately from each parent. An allele can be in different allelic states. Page 6.

Allelic state: A specific gene can be in different allelic states. Page 6.

Base pair: A base pair is constructed from two nitrogenous bases of the four A, T, C and G. A always pair with T and C always pair with G. Page 5.

Chromatide: A double helix DNA strand, humans have 92 chromatides in all. Page 5.

Chromosome: Humans have 23 pairs of chromosomes. Each chromosome consists of two chromatides. Page 5.

Codominant: When both alleles at a locus is phenotypically expressed. Page 7.

Codon: A codon is constituted of 3 base pairs and can either specify an amino acid or a stop code in the coding of genes. Page 5.

Crossover: The event where genetic material exchange between two chromosomes that pair during meiosis. Page 9.

DNA Strand: See chromatide. Page 5.

DNA: Abbreviation for Deoxyribonucleic acid. The molecular structure that encodes genetic information. Page 5.

Dominant: An Allele is dominant over another allele if the other allele is recessive (at a given locus). If the allelic state of the allele is not expressed the allele is recessive. Page 7.

Expressed: An allele is expressed if it is expressed in the phenotype. Page 7.

Gene: The sequence of codons that encode a single protein. Page 6.

Genotype: Refers to the allelic states of the two alleles at some locus. These states constitute a persons genotype at that locus. Page 7.

Heterozygous: An individual is heterozygous if the maternal and paternal gene at a specific locus are different. Page 7.

Homozygous: An individual is homozygous if the maternal and paternal gene at a specific locus are the same. Page 6.

IBD: Identical By Descent. Two individuals who have inherited a common allele. Page 15.

IBS: Identical By State. Two individuals with the same allelic state for some marker. The alleles need not be identical by descent. Page 15.

Interference: Occurrence of one crossover highly reduces the likelihood of another in its vicinity. Page 10.

LOD score: Likelihood of Odds (LOD) is a scoring function which determines the likelihood of two loci being linked given a recombination fraction versus the likelihood that they are unlinked. Page 14.

Linked: Denotes that the recombination is less than $\frac{1}{2}$ between two loci. Page 10.

Locus: The position on the chromosome where a certain gene is located. Page 6.

Map function: The function that specifies the relationship between physical distance and recombination fraction. Page 10.

Marker: An identifiable physical location on a chromosome whose inheritance can be observed. Page 6.

Maternal: Refers to the allele received from the mother at a specific locus. Page 6.

Meiosis: A cell division process where haploid cells are divided to form diploid germ cells. In our context some of the cells are later combined with other germ cells to form the genome of a new individual. Page 7.

Mendelian inheritance: In our context, the two laws that influence the pattern of inheritance for genes. Page 11.

Morgan: Denotes a distance where one recombination can be expected on average. Page 10.

Paternal: Refers to the allele received from the father at a specific locus. Page 6.

Phase: Refers to the relationship between two alleles at different loci. If they are in phase they are residing on the same chromosome. Page 9.

Phenotype: The observable characteristic of an individual. Page 7.

Protein: Important molecules in the human body that influence many traits. Page 6.

Recessive: When an allele is only phenotypically expressed in the homozygous state. Page 7.

Recombination fraction: The probability that an odd number of recombinations occur between two loci. Page 9.

Recombination: The process by which offspring receives a combination of linked genes different from that of either parent. This occurs by a crossover between parental chromatides during meiosis. Page 9.

Scoring Function: A measure of resemblance between inheritance patterns of a trait and a marker. Page 14.

Strand: See chromatide. Page 5.

Trait: An observable attribute of an individual. Page 6.

Triplet: See codon. Page 5.

Unlinked: Denotes that the recombination is $\frac{1}{2}$ between two loci. Page 10.

bp: See Base pair. Page 5.