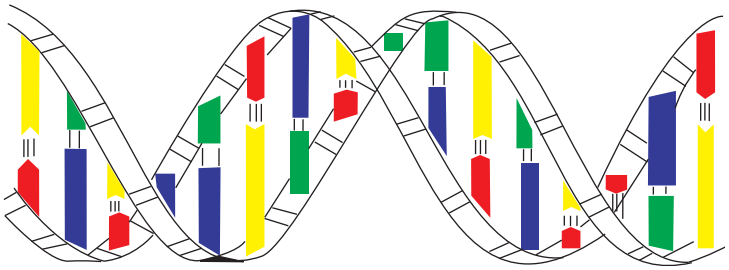
Linkage Analysis with PDGs



Group E1-117a, DAT6

Aalborg Universitet
Department of Computer Science

8th of June 2004

Mads D. Thrane
Mette Thøgersen

Faculty of Engineering and Science
Aalborg University

Department of Computer Science, Fredrik Bajersvej 7E, DK-9220 Aalborg

TITLE:

Linkage Analysis with PDGs

TOPIC:

Linkage Analysis,
Graphical Representation,
Probabilistic Approaches, Probabilistic Decision Graph.

PROJECT PERIOD:

DAT5-6,
September 1st, 2003 -
June 8th, 2004

PROJECT GROUP:

E1-117a

GROUP MEMBERS:

Mads D. Thrane
Mette Thøgersen

SUPERVISOR:

Manfred Jaeger

NUMBER OF COPIES: 7 (+ 1 online)

REPORT PAGES: 84

APPENDIX PAGES: 16

TOTAL PAGES: 104

SYNOPSIS:

This is a linkage analysis project. Linkage analysis is a tool for locating genes on DNA strings. The motivation has been to investigate possible optimization of an existing linkage analysis algorithm FASTTREETRAVERSAL, developed by a medical company DeCode Genetics in Iceland, using probabilistic graphical models. The current implementation of the FASTTREETRAVERSAL Algorithm uses MTBDDs.
The project is divided into three parts. The first part is an introduction to the field of linkage analysis, written to create a better understanding of the subject. The second part is an investigation into some of the currently available linkage analysis algorithms, and the third part is development of a new linkage analysis algorithm using PDGs.
We have implemented a single point linkage analysis algorithm which gives an RFG as output. To use the output as input to a multi point algorithm the RFG must be normalized into a PDG. The implementation has been tested using data from the Superlink homepage. Superlink is another linkage analysis algorithm, which uses probabilistic graphical models, in this case Bayesian networks. Giving this data to the implementation the resulting RFG contains 105 nodes, which is quite small compared to the possible 4^{42} nodes.

Contents

1 Introduction

The motivation behind this project originates form a question asked by Anna Ingolfsdóttír, Associate Professor at Aalborg University, and associated with BRICS (Basic Research in Computer Science). She asked us to investigate whether Bayesian networks or other graphical models might be useful in linkage analysis. To answer this question we first had to establish an understanding of what linkage analysis is and the algorithms currently in use.

We chose to apply Probabilistic Decision Graphs (PDG) to the linkage analysis algorithm. PDGs were chosen because the current algorithms indicate that the two best data structures are Bayesian networks and Binary Decision Diagrams (BDD), and PDGs as a data structure give us a combination of the best features of these.

1.1 Collaboration

This project was made in collaboration with DeCode Genetics of Iceland. DeCode is a pharmaceutical company. They use linkage analysis as a part of the process of locating disease genes for inherited diseases.

DeCode works with a network of doctors in Iceland, and through this network they have access to the genetic material of the living Icelandic population. The population of Iceland is particularly suited for doing linkage analysis because:

1. The church records hold a full account of family relations, dating all the way back to the 9th century.
2. It is a very homogeneous population, where very few new individuals have been added over the generations. This means that the col-

lective genetic material is a constant factor, and it is easier to estimate the frequency of traits over the population.

When the doctors in the network come in contact with a patient with a specific trait or disease currently under investigation, blood samples are collected. Both from the patient and the living relatives of the patient. These blood samples are sent to DeCode together with information on the family relations[1]. DeCode then analyzes the blood samples using biological methods, to get the information needed for performing genetic analysis.

Through linkage analysis possible areas of location[2], for the trait under investigation, are located for further investigation. The located areas are still quite large, with respect to the size of genes, and a more detailed search method is applied to find the exact location of the gene. This method is called *association*.

Association compares the segments found by linkage analysis, one little DNA sequence at a time. The DNA on these segments encode different not yet identified genes. The segments stem from several people who has the trait under investigation, and in association they search for identical DNA sequences. In families the DNA is very similar, and therefor locating the genetic code for a specific trait is easier the more the rest of the DNA differs. Because the genes are compared in pairs using biological means of investigation, the method of association is slow and expensive. Linkage analysis is faster and cheaper, and is therefor an important tool to narrow down the segment of DNA which will undergo closer investigation.

Finally when a promising gene has been found, it is examined by bombarding it with different proteins to discover the behavior of the gene, which in turn helps to create medicine that cure the given trait.

1.2 Report Structure

The report consists roughly of three parts: First part is the fundamental theory needed for understanding what linkage analysis is. This includes an introduction to human genetics in Chapter 2, and the definition of linkage analysis in Chapter 3. The second part is an investigation into the existing algorithms for linkage analysis going into the details of four of the algorithms in Chapter 4, and describing the differences between these by direction of inference in Chapter 5. The third and final part is design and implementation of a new linkage analysis algorithm using PDGs. In Chapter 6 PDGs are defined and the operations needed for doing linkage

[1] Pedigree information.
[2] Segments of DNA of a chromosome.

analysis are described. The new linkage analysis algorithms are discussed in Chapter 7, and some of the details of the implementation of the single point algorithm is given in Chapter 8.

2 Human Genetics

The following is a brief introduction to the very basics in human genetics. To delve further into the subject of human genetics with respect to genetic analysis see [17].

Human beings are diploids meaning that every individual carries two copies of each chromosome, such that each cell in the human body contains 23 chromosome pairs[1].

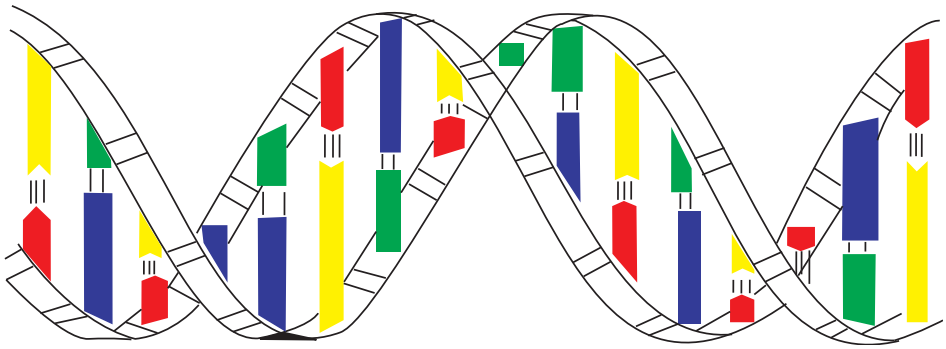


Figure 2.1: A small part of a DNA string. (A = blue, T = green, G = yellow, C = red)

A *chromosome* is a long string of DNA (deoxyribonucleic acid). DNA is a double helix molecule with two sugar-phosphate backbones and four nitrogenous bases: adenine (A), thymine (T), cytosine (C) and guanine (G). These bases fit together two and two in base pairs (bps): (A,T), (C,G) and thereby bind the two backbones together, see Figure 2.1. Three base pairs encode an amino acid or a stop code. A sequence of amino acids form a protein molecule, which we term a *gene*[2]. Each chromosome contains a

[1] 22 autosomes, and 1 pair of sex chromosomes.

[2] There are some discussions into the correct usage of the word gene.

large number of genes.

Each pair of chromosomes consist of one *maternal* and one *paternal* chromosome. The maternal chromosome of an individual is derived from the DNA of his or her mother. The paternal is derived from the father.

Sex cells (sperm- and egg cells) consist of only 23 single chromosomes, one for each pair in the parent. During the creation of the sex cells, the two chromosomes of a pair mix in a process called meiosis, such that the resulting chromosome consist of segments of DNA from each chromosome in the pair, see Figure 2.2. Where DNA from one chromosome is inserted in the DNA of the other, we say that a *crossover* has occurred.

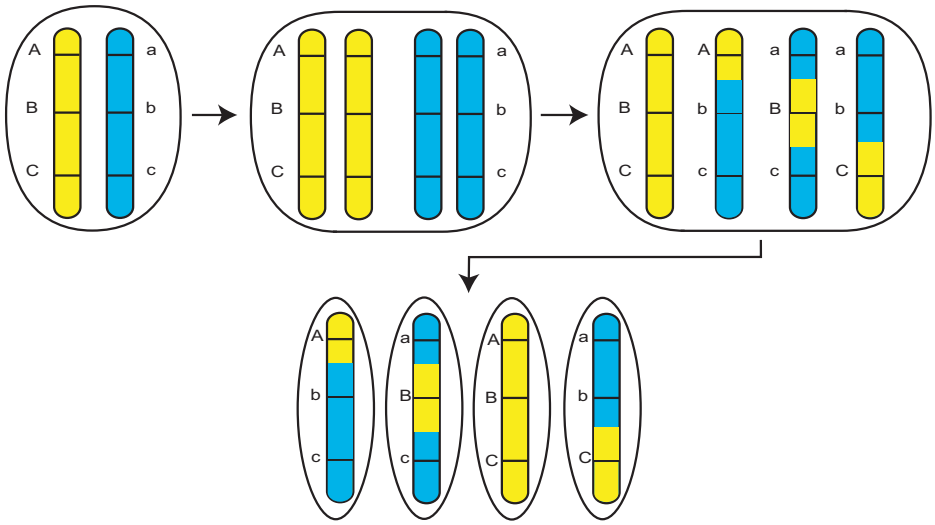


Figure 2.2: The single chromosome passed to an individual from a parent can be build from different pieces of the two chromosomes of the original pair of the parent.

The position of a gene on a chromosome pair is called the *locus* of the gene. The genetic distance between two loci is defined as the expected number of crossovers taking place in a single meiosis between those two loci. The unit of genetic distance[3] is called a *Morgan*, which is the average number of crossovers, [3].

Different variants of DNA that can be assumed at a locus, are called *alleles*. The pair of alleles at any locus is known as the *genotype* of that locus. If both alleles at the locus are of the same type, the genotype is said to be *homozygous*. If they are of different type, it is said to be *heterozygous*.

A *phenotype* is the observable characteristic of a gene. The difference between phenotype and genotype stems in part from that alleles can be *dominant*, *co-dominant* or *recessive*.

[3]Note! Genetic distance is *not* the same as physical distance.

A genetic trait for which the expressed phenotype corresponds to the genotype at a single locus is often called a *Mendelian* or single-locus trait. The human ABO blood group is an example of such a trait (ignoring the Rhesus factor), see Table 2.1. When talking about a blood type of a person, we actually refer to the phenotype.

m \ p	**A**	**B**	**0**
A	A	AB	A
B	AB	B	B
0	A	B	0

Table 2.1: The top row is the allele located on the paternal chromosome. The left most column is the allele located on the maternal chromosome. The table holds the phenotypes given the combined alleles.

If a person is of blood type A, the actual genotype will be either AA or A0. This is because A and B are dominant over 0. Thereby is 0 a recessive gene, which only is observable in blood type 0 of an individual who is of homozygous genotype (00). A and B are co-dominant to each other, i.e. non of them suppress the other.

Some traits has a phenotype, which is affected by the simultaneous segregation of many genes at many loci, i.e. the encoding of the trait is shared between several loci. These are called *quantitative* traits. They may in addition have some non-genetic variation superimposed, i.e. the underlying genotype effects on the trait phenotype may vary with age and sex and various environmental factors. Quantitative traits can exhibit variation on a continuous scale, but can also be discrete as in threshold traits. A quantitative trait locus can be thought of as a segment of chromosome affecting a quantitative trait but whose effect is not large enough to cause an observable discontinuity and is hence not detectable using Mendelian methods.

Summary

In this chapter we have given a very brief introduction to human genetics. In short we have found that:

- Human beings are diploids meaning that every individual carries two copies of each chromosome, such that each cell in the human body contains 23 chromosome pairs.
- A chromosome is a long string of DNA. DNA is a long string of genes.
- Each pair of chromosomes consist of one *maternal* and one *paternal* chromosome.

- Sex cells (sperm- and egg cells) consist of only 23 single chromosomes, one for each pair in the parent.
- During the creation of the sex cells, the chromosomes mix in meiosis, such that the resulting chromosome consist of segments of DNA from each chromosome in the pair. Such a switch is called a *crossover*.
- The position of a gene on a chromosome pair is called the *locus* of the gene.
- The genetic distance between two loci is defined as the expected number of crossovers to occur in a single meiosis between the two loci.
- Different variants of DNA that can be assumed at a locus, are called *alleles*.
- The pair of alleles at any locus is known as the *genotype* of that locus.
  - If both alleles at the locus are of the same type, the genotype is said to be *homozygous*.
  - If they are of different type, it is said to be *heterozygous*.
- A *phenotype* is the observable characteristic of a gene, where alleles can be:
  - dominant,
  - co-dominant or
  - recessive.
- A genetic trait for which the expressed phenotype corresponds to the genotype at a single locus is often called a *Mendelian* or single-locus trait.
- A phenotype which is affected by the simultaneous segregation of many genes at many loci, i.e. the encoding of the trait is shared between several loci, are called *quantitative* traits.

3 Linkage Analysis

In genetic linkage studies, the aim is to locate the genes for some trait of interest by mapping their positions relative to known marker loci within the pedigrees being studied[1].

Traits for which the locus, all alleles and their population frequencies are known are called *markers* in linkage analysis. The *allele frequency* in a population is given as the percentage of the population, which has this allele.

Definition 1 *A marker can be defined as* $M = \langle l, \mathbf{A}, \pi(\mathbf{A}) \rangle$ *where:*

- l *is the locus of a gene.*
- $\mathbf{A}$ *is the set of all possible alleles of a gene.*
- $\pi(\mathbf{A})$ *is the frequency of the alleles over the population.*

The idea behind linkage analysis is to compare the inheritance pattern of the trait under investigation to inheritance patterns at the markers. If two inheritance patterns are very similar, there is a high probability of the genes being located close to each other. If two genes are located close to each other they are said to be linked, hence the name *linkage analysis*.

Linkage analysis is performed on a *pedigree*, which is a group of individuals together with a full specification of all the familial relationships between them, see Figure 3.1. A pair of pedigree members are defined to be *spouses* only if they have mutual offspring in the pedigree and every such pairing is called a *marriage*. The individuals without parents in the pedigree are called *founders* of the pedigree and these, by definition, are unrelated. Those with parents in the pedigree are called *non-founders*.

[1] Marker loci are assumed to have no effect on the trait under consideration.

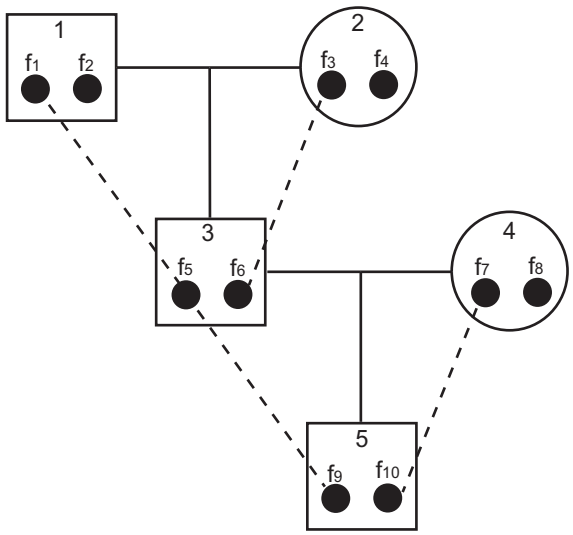


Figure 3.1: A example pedigree. Squares are males, circles are females. The left allele is the paternal allele, the right the maternal allele. The dotted lines depicts an example inheritance pattern, in this case every one has inherited the paternal alleles of their parents.

Definition 2 *A pedigree is* $P = \langle \mathbf{I}, \mathbf{F}, \mathbf{E} \rangle$ *where the following hold:*

- $\mathbf{I}$ *is the set of individuals in the pedigree.*
- $\mathbf{F}$ *is the set of individuals with no parents in the pedigree,* $\mathbf{F} \subseteq \mathbf{I}$*.*
  - $\mathbf{N} = \mathbf{I} \backslash \{\mathbf{F}\}$ *is the set of non-founders.*
- $\mathbf{E}$ *is the set of family relations between the individuals, which are defined such that*
  - *No one can be their own ancestor.*
  - *No one can be both a mother and a father.*

An *inheritance pattern* describes how the genes have been passed from generation to generation, down through the pedigree. See Figure 3.1.

In reality there is one *true* inheritance pattern for each pedigree at each marker. This cannot directly be read from the biological examinations however, so in linkage analysis we deduce a set of *possible* or *compatible* patterns, and of these consider the most probable pattern to be the true pattern. This means that in linkage analysis we work with sets of inheritance patterns of a marker in a pedigree.

In Figure 3.2 the same pedigree with the same genotype information is given, resulting in two possible inheritance patterns.

Figure 3.2: Two possible inheritance patterns of a small pedigree.

Inheritance patterns for two different loci indicate whether the alleles come from the same grandparent, if so they are said to be *in phase*. In Figure 3.3 a cross over has occurred from individual 3 to individual 5, the two paternal alleles of individual 5 are therefore not in phase.

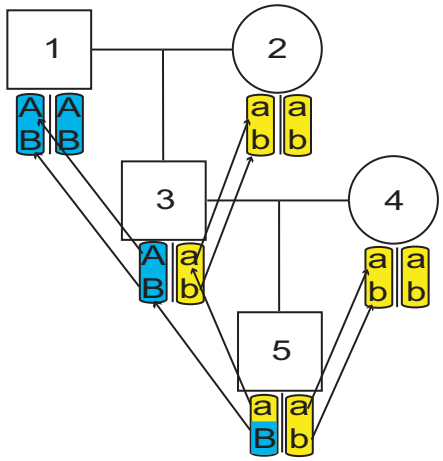


Figure 3.3: An example inheritance pattern is given for each of the two loci. A cross over *must* have occurred for the paternal chromosome of individual 5.

The given genotype information is *unordered*. This means that given the information that a person is of genotype Aa in Figure 3.2, it is not possible to determine which parent provided the A allele and which provided the a, until a specific inheritance pattern is deduced.

If the most probable inheritance patterns of two loci are very similar, few crossovers are assumed to have occurred, which could indicate that the alleles are in linkage.

Establishing the number of crossovers between two loci is very difficult, because crossovers generally cannot be observed. The only way to measure crossovers is to observe the differences in the inheritance patterns of known markers. In Figure 3.3 more crossovers might have occurred, than the ones that can be observed directly in the graph. It is not possible to

detect crossovers which have occurred for homozygous genes, or to differentiate between one crossover occurring or any other odd number of crossovers. It is also not possible to differentiate between no crossover occurring and any even number of crossover occurring, see Figure 3.4.

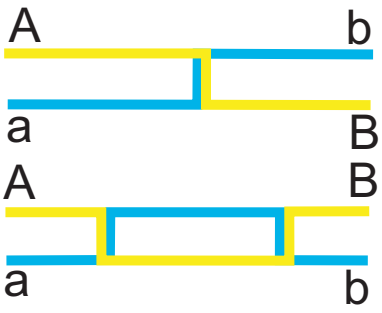


Figure 3.4: It is only observable when an *odd* number of crossovers have occurred between two markers.

In linkage analysis an odd number of crossovers is called a *recombination*, and an even number is no recombination.

$$\#crossover = \begin{cases} odd & = & \text{recombination} \\ even\ or\ 0 & = & \text{no recombination} \end{cases}$$

The genetic distance between two loci is in linkage analysis defined to be the probability of a recombination occurring between the loci. This is called the *recombination fraction* or *recombination frequency*, [13].

A generally applied equation for calculating the relationship between the distance in Morgans[2] d between two loci and the recombination frequency θ is:

$$d = -\frac{1}{2}ln(1-2\theta) \tag{3.1}$$

$$\theta = \frac{1}{2}(1-e^{-2d}) \tag{3.2}$$

Where $0 \leq \theta \leq \frac{1}{2}$. If $\theta \approx 0$ the loci are very close to each other, and will tend to be in phase[3]. If $\theta = \frac{1}{2}$ the two loci are said to be unlinked, independent of each other. Possibly even on different chromosomes[17].

3.1 Two Approaches to Linkage Analysis

There are generally two classes of approaches to linkage analysis: *single point* and *multi point* linkage analysis. In single point analysis only one

[2]The mean number of crossovers.
[3]Inherited from the same grandparent.

marker is investigated at a time, independently of other markers, and the resulting most probable inheritance pattern is compared to the inheritance pattern of the trait. In multi point analysis the probability distributions of the inheritance patterns deduced on adjacent markers influence each other dependent on the genetic distance between said markers, see Figure 3.5.

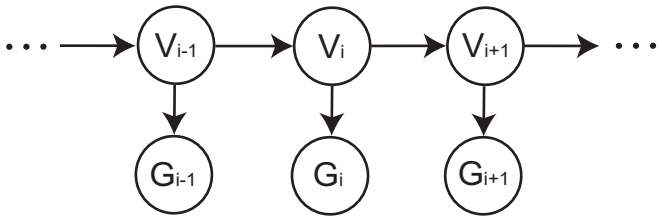


Figure 3.5: The probability distributions at the different loci influence each other. V_i is the set of inheritance patterns of marker M_i, and G_i is the genotype information given for M_i.

If two loci are close, some of the inheritance patterns at these loci might be identical. These inheritance patterns are more probable than the other inheritance patterns at the given loci. This is due to the fact that the closer the two loci are, the lower is the number of expected recombinations.

The genetic distance between two markers is a important factor when doing linkage analysis, especially multi point analysis. The distance between any two markers is recorded in a genetic map[4]. The genetic map describes the distance between the different markers by estimation of the amount of recombinations occurring between the neighboring markers. When a gene has been located and all possible alleles identified, this information combines into a new marker, and the genetic map is expanded.

Definition 3 *A genetic map of a chromosome can be defined as* $\mathcal{G}_{map}(\mathcal{C}) = \langle \mathbf{M}, \Theta \rangle$, *where*

- $\mathcal{C}$ *is a chromosome,*
- $\mathbf{M}$ *is the set of markers on* $\mathcal{C}$*, and*
- Θ *is the set of recombination fractions, where* θ_i *is the recombination fraction between* M_i *and* M_{i+1}.

The genetic maps can be faulty in several ways [20]. Markers might be in the wrong place on the map, resulting in linkage analysis indicating areas of interest, which in fact are not interesting at all, because the estimated recombination fraction between two markers is based on the assumption that these two markers in fact are located next to each other. To complicate

[4]Maps over chromosomes.

matters even more, for different parts of the population, markers can be placed at different loci of the chromosome.

Another error in the genetic map is the possibility of additional (unknown) alleles at markers. This means that the population frequencies at such a marker are wrong. If there are high discrepancy in the estimation of the allele frequency, this can have an impact on the probability calculation of inheritance patterns.

In this report we will however focus only on the linkage analysis problem areas, and save the problems of the genetic map to other projects.

3.2 Measuring Linkage

When the most probable inheritance pattern for a given locus is discovered, the distance between the locus and the trait is estimated.

One method for calculating linkage distances is called the LOD Score Method developed by Newton E. Morton. This method is described in [30] and [6] and is the most widely used method for calculating the linkage distance. Calculating the LOD score is an iterative process, where a series of LOD scores are calculated from a number of proposed linkage distances between two loci. The highest LOD score is considered to be the correct linkage distance estimate.

$$LOD = log\left(\frac{\mathcal{P}(v_i|\theta < 0.5, v_t)}{\mathcal{P}(v_i|\theta = 0.5, v_t)}\right) \tag{3.3}$$

where v_i is the most probable inheritance pattern for a locus, and v_t is a proposed inheritance pattern of the trait. In effect what is counted is the number of recombinations which have occurred between v_i and v_t.

Example: Consider inheritance patterns of the two genes in Figure 3.6. In most of the individuals of the pedigree, it seems that the two alleles A and X move as a block when passed to the children. In individual 9 however a recombination has occurred and A now resides on the same chromosome as x. (In the example we only look at the inheritance pattern of the 3rd generation, because recombination cannot be observed in the higher generations.)

If the first estimate of the recombination fraction (linkage distance) is 0.125. The probability of no recombination occurring is then $(1 - 0.125)$.

In the 3rd generation of the pedigree there is a total of seven individuals where no recombination seems to have occurred, and one individual with a recombination, so the total probability of the inheritance patterns based on a recombination frequency of 0.125 is given by:

$$(0.875)^7(0.125)^1 = 0.0491$$

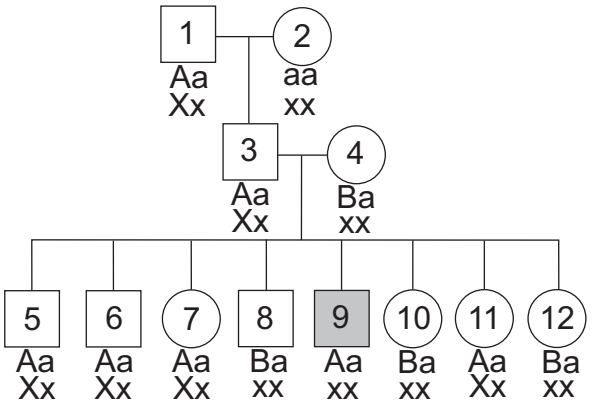


Figure 3.6: An example pedigree, showing the inheritance of two genes. The pedigree implies linkage between the two, with recombination occurring only for individual 9.

The probability of the unlinked inheritance patterns are then:

$$(0.50)^8 = 0.00391$$

The LOD score is then the logarithmic value of the linkage probability divided by the independent probability:

$$LOD_{0.125} = log\left(\frac{0.0491}{0.00391}\right) = log(12.566) = 1.099$$

This calculation is repeated for a series of recombination frequency estimates, and the largest LOD score is picked to be the distance between the two genes.

A LOD score of 1 indicates that the likelihood of linkage occurring is 10 times the likelihood of no linkage. If the score is 2 then it is a 100 times bigger. In practice a LOD score is preferred to be higher than 3, which means that the likelihood of linkage occurring at the given estimated distance is 1000 times greater than that of no linkage.

3.3 Definition of Linkage Analysis

The approaches to linkage analysis are many and diverse, see Appendix A. We therefore create a common definition to pinpoint the similarities and diversities of the different approaches.

Linkage analysis is based on analyzing the existing genotype information given for some or all of the people in the pedigree [5].

[5]Sometimes phenotype information is given, this can be seen as partial genotype information.

Definition 4 *Genotype information available for all individuals in the pedigree can be defined as* $\mathcal{G}(\mathbf{M})$ *where:*

- $\mathbf{M}$ *is a set of markers,*
- $\mathcal{G}_i(\mathbf{M})$*, the genotype information available for individual* i*, is given by*
  - $\mathcal{G}_i(\mathbf{M}) = \{(\alpha_{1_1}, \alpha_{1_2}), \ldots, (\alpha_{m_1}, \alpha_{m_2})\}$ *where each* $(\alpha_{j_1}, \alpha_{j_2})$ *is either the unordered genotype of marker* j*, or* $(x_{j1}, x_{j2}) = (\emptyset, \emptyset)$ *representing that no genotype information is available for marker* j*.*

So far we have described linkage analysis as three steps:

1. The probabilistic step, where the probability distribution for the inheritance patterns of the markers is found.
2. The LOD score calculation, where linkage with respect to the trait under investigation is calculated.
3. The evaluation step, where the most probable areas of the DNA string are picked for further analysis.

Note: From here on when we are talking about linkage analysis we mean the probabilistic step in linkage analysis.

We thereby define linkage analysis as a function of a pedigree, a set of markers and genotype information given on the markers for the pedigree.

Definition 5 *Linkage analysis can be seen as a function* $\mathcal{L}(P, \mathcal{G}(\mathbf{M}))$*, which output* $\mathcal{P}(\mathbf{v}_{\mathbf{M_i}}|\mathcal{G}(\mathbf{M}))$ *where:*

- P *is a pedigree*
- $\mathcal{G}(\mathbf{M})$ *is the genotype information available for set of markers under investigation* $\mathbf{M}$*, for the individuals in* P*.*
- $\mathcal{P}(\mathbf{v}_{\mathbf{M_i}}|\mathcal{G}(\mathbf{M}))$ *is the probability distribution of the set of inheritance patterns for each marker given the genotype information.*

In reality the genotype information given is filled with holes, both because the biological methods for extracting the information can smudge, but mainly because the methods for gaining the information is quite new, and therefor only the younger generations are available for genotyping.

Summary

In this chapter we have given a introduction to linkage analysis. In short we have found that:

- Linkage analysis is investigation of the inheritance patterns for given markers in a pedigree, and comparing these to the inheritance pattern of the trait under investigation.
- A *marker* is a locus where all the possible alleles, the population frequencies of these and the biological function of the gene are known.
- A *pedigree* is a group of individuals together with a full specification of all the familial relationships between them.
  - Those without parents in the pedigree are called *founders* of the pedigree and these, by definition, are unrelated. Those with parents in the pedigree are called *non-founders*.
- An *inheritance pattern* describes how the genes have been passed from generation to generation, down through the pedigree.
- If two inheritance patterns are very similar, there is a high probability of the genes being located close to each other. If two genes are located close to each other they are said to be *linked*.
- The *true* inheritance pattern cannot be directly read from the biological examinations however, so in linkage analysis we deduce a set of *possible* patterns, and of these consider the most *probable* pattern to be the true pattern.
- If two inheritance patterns of two loci are very similar, few crossovers are assumed to have occurred, which could indicate that the alleles are linked.
- In linkage analysis an odd number of crossovers is defined to be a recombination, and an even number to be no recombination.
- The genetic distance between two loci is in linkage analysis defined to be the probability of a recombination occurring between the loci. This is called the *recombination fraction*.
- There are generally two classes of approaches to linkage analysis: *single point linkage analysis* and *multi point linkage analysis*.
  - In single point analysis only one marker is investigated at a time, independently of other markers, and the resulting inheritance patterns are compared to the inheritance pattern of the trait.

  - In multi point analysis the information given on adjacent markers influence each other dependent on the genetic distance between said markers.

- Linkage analysis can be seen as three steps:

  1. The probabilistic step, where the probability distribution for the inheritance patterns of the markers is found.
  2. The LOD score calculation, where the linkage with respect to the trait under investigation is calculated.
  3. The evaluation step, where the most probable areas of the DNA string are picked for further analysis.

  From here on when we are talking about linkage analysis we mean the probabilistic step in linkage analysis.

- We define linkage analysis as a function of a pedigree, a set of markers and genotype information given on the markers for the pedigree: $\mathcal{L}(P, \mathcal{G}(\mathbf{M}))$.

4 Algorithms for Linkage Analysis

There are many different algorithms and tools, which have been developed for doing linkage analysis using computers. See Appendix A for a list of the most popular publicly available tools. We have chosen four linkage analysis approaches for scrutiny and comparison:

- the Elston-Stewart Algorithm [8], which was written in 1971 and the ideas of which many linkage analysis tools have been developed,
- the Lander-Green Algorithm [23][21], which is one of the fundamental algorithms specifically for linkage analysis,
- FastTreeTraversal which is based on the ideas from Lander-Green, and implemented in the Allegro software package developed by DeCode [16], and
- Superlink [9] which utilized Bayesian networks.

First we introduce a small example pedigree. Then each approach will be described and applied to the example for clarification.

4.1 A Small Example Pedigree

The example pedigree in Figure 4.1 consist of three generations of in all 8 individuals. Four founders (two top-level and two spouses in the second generation) and four non-founders.

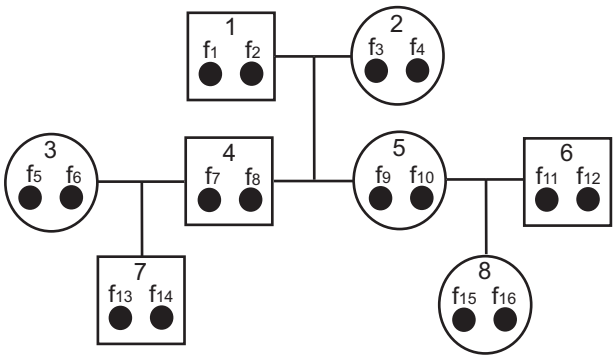


Figure 4.1: An example pedigree of three generations, which will be run through the different approaches for clarification purposes. In reality of course the pedigree is too small to provide any reliable information.

The example include a set of markers $\mathbf{M} = \{M_1, M_2\}$ with two possible alleles per marker:

- the set of possible alleles $\mathbf{A_1} = \{a, A\}$ for M_1, with population frequency $\pi(a = .75, A = .25)$ and
- the set of possible alleles $\mathbf{A_2} = \{b, B\}$ for M_2, with population frequency $\pi(b = .95, B = .05)$.

We have genotype information given for all the individuals of the two lower generations of the pedigree, see Table 4.1. The two founders in the topmost generation have no genotype information. This is to illustrate that usually these are not genotyped due to the fact that founders are usually long dead.

I:	**3**	**4**	**5**	**6**	**7**	**8**
$\mathbf{M}_1$	aa	Aa	aa	AA	aa	Aa
$\mathbf{M}_2$	bb	Bb	bb	bb	Bb	bb

Table 4.1: The unordered genotype information given for the example pedigree.

4.2 The Elston-Stewart Approach

In 1971 R. C. Elston and J. Stewart [8] developed an approach for finding the likelihood of genotypes of pedigree data based on population distribution of genotypes, phenotype distribution for the different genotypes, and offspring distribution given the genotype of the parents. The approach is based on their backgrounds in genetics and statistics and therefor need a bit

of restructuring to be compared to the approaches developed by computer scientists.

The approach was not developed for linkage analysis, however the architecture and ideas developed in this approach have been the basis of many genetic analysis algorithms developed at a later time. In the following we will apply Bayesian networks to the approach to provide better understanding. For an introduction to Bayesian networks see Appendix B.

The Elston-Stewart algorithm calculates the likelihood for one nuclear family at a time. The likelihood of a single sibship of n individuals with phenotypes $x_1, \ldots, x_n$, given that the parents have genotype s and t, and the probability of an individual having genotype u (where s, t and u can be of values $1, 2, \ldots, k$) is in [8] given by equation 4.1.

$$\prod_{i=1}^{n} \sum_{u=1}^{k} p_{stu} g_u(x_i) \tag{4.1}$$

We translate equation 4.1 to Bayesian terminology, one variable at a time. $g_u(x_i)$ is the probability of a phenotype trait given the genotype, $\mathcal{P}(x = i|u)$, where x might be a quantitative trait, and the genotype in that case is given as a segregation of several loci. If x is a single loci trait such as blood type, and u is the genotype, then

$$g_u(x_i) = \begin{cases} 1 & \text{if } x_i \text{ is determined by } u \\ 0 & \text{otherwise} \end{cases}$$

Moving on in equation 4.1, p_{stu} is the probability of the genotype of individual u given the parents s, t. This is in Bayesian terms written $\mathcal{P}(u|s, t)$.

Likelihood computation of founder genotypes is based on population frequency. Elston-Stewart introduces the probability ψ_v of a given founder being of the v-th genotype, i.e. ψ_v is the proportion of individuals in the population who have the v-th genotype. The likelihood of observing a founder being of a specific genotype, is in Elston-Stewart then given by equation 4.2.

$$\sum_{v=1}^{k} \psi_v g_v(x_i) \tag{4.2}$$

In Bayesian networks the probability distribution is $\mathcal{P}(S)$ where each entry in the probability table is given by:

$$\mathcal{P}(S = v) = \psi_v$$

Summing up the probability distribution of each nuclear family is given by:

- $\mathcal{P}(\mathcal{G}_y)$ the probability of the genotype of a founder y, based on population frequency,
- $\mathcal{P}(Ph_n|\mathcal{G}_n)$ the conditional probability distribution of the phenotype of individual n given the genotype of n, and
- $\mathcal{P}(\mathcal{G}_n|\mathcal{G}_m, \mathcal{G}_p)$ the conditional probability distribution of a genotype of a child n given the genotypes of the parents m, p.

Together all these the probabilistic elements for a single nuclear family can be represented in a graphical manner as seen in Figure 4.2, giving a Bayesian network.

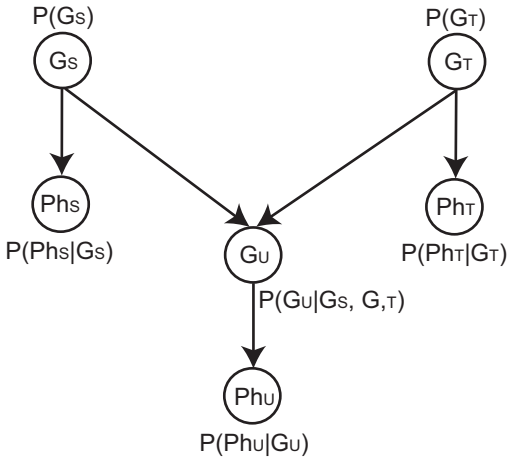


Figure 4.2: A graphical representation of the Elston-Stewart probability distributions given for a single child and its parents. Together with the probability distributions of each variable this is a Bayesian network, see Appendix B.

In Elston-Stewart the genotype of an individual is seen as a string of information, which can represent several markers or traits. This means that the length of the genotype information for each individual is two times the number of markers under investigation. The number of possible genotype configurations under investigation grow exponentially in the number of markers, and thereby so does the probability table for each node in the Bayesian network. It means however that whenever the approach operates on a variable for an individual, it works across all the markers of that one individual at the same time. Thereby is the complexity of the algorithm linear in the number of people in the pedigree and exponential in the number of markers.

4.2.1 Elston-Stewart Example

In Elston-Stewart the most natural graphical description of a pedigree is a relationship graph [28], which for the example given in Section 4.1 can be seen in Figure 4.3. The pedigree is translated into the genotype graph given in Figure 4.4.

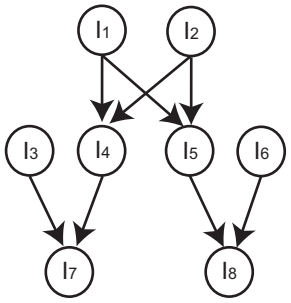


Figure 4.3: In Elston-Stewart the most appropriate description of a pedigree is given using a relationship graph.

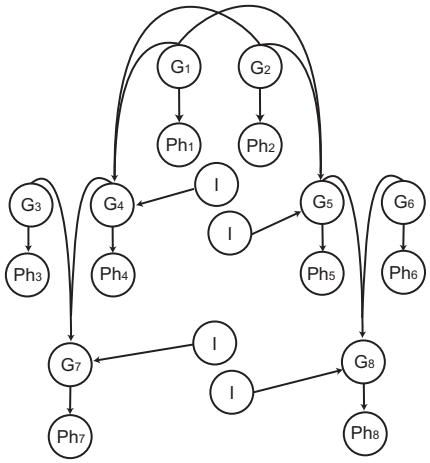


Figure 4.4: An example of a Bayesian network describing the probabilistic dependability of the genotype and phenotype of the specific individuals, and between the genotypes of the three generations in the pedigree of Figure 4.3.

When running the example we ignore the phenotype nodes, because the given information is genotype information, thereby either totally separating the phenotype nodes from the rest of the network, or leaving the phenotype nodes barren.

Each founder genotype probability table will consist of nine probability entries, one for each detectable genotype combination of the two markers. The genotype information is unordered, and therefor the genotypes $\{AaBB\}$ and $\{aABB\}$ are considered to be the same. The non-founder genotype probability tables will be consist of 9^3 entries. One for each the probability of each possible genotype of the child given each possible genotype combination at the two parents.

For the evidence given many configurations become impossible, i.e. are set to probability zero. Elston-Stewart never takes these into account and we model this by reducing the tables by removing all columns and rows for impossible values. For each impossible genotype combination in a parent, an entire column is impossible for the child.

When the genotype information from Table 4.1 has been entered as evi-

dence the probability tables are reduced to size 1 for each of the two genotyped founders and the non-founders of 3rd generation, and size 9^2 (an entire row) for each of the non-founders of 2nd generation. The probability tables of the two 1st generation founders are only reduced by removal of a single entry $\{AABB\}$. This entry is impossible because of the genotype of individual 5. Entering the given genotype information thereby reduces the total size of the probability tables from 2953 to 164 entries.

4.3 The Lander-Green Algorithm

Lander-Green takes a somewhat different approach to linkage analysis. Where Elston-Stewart peels one nuclear family at a time for all markers on the chromosome, Lander-Green peels one entire pedigree for a single marker at a time. Actually Lander-Green starts by doing single point analysis for each marker, and then proceeds to update the probability distribution of inheritance patterns at one marker with respect to the neighboring markers, as described in Section 3.1.

Lander-Green encodes inheritance patterns as binary vectors, where each bit denotes one inherited allele of a non-founder, and the value of each bit describes whether the inherited allele is the maternal or paternal allele of the parent.

Definition 6 *Inheritance vector v_i of individual i is a pair of bits (b_1, b_2), where b_1 is the paternally inherited allele of i, and b_2 is the maternally inherited allele of i. The value of the bits correspond to the paternal or maternal allele of the parent, p and m respectively.*

As an example consider the inheritance vector $v_i = (m, p)$ for individual i. This means that individual i inherited one allele from its father's mother, and one allele from its mother's father.

The total inheritance vector v of a pedigree is a concatenation of bit pairs for each non-founder. This means that for a pedigree of n non-founders the inheritance vector is $2n$ long.

Algorithm LANDER-GREEN($P, \mathcal{G}(\mathbf{M})$)
1. **for** each $m \in \mathbf{M}$
2. **do** $(\mathcal{A},\mathcal{U},\mathcal{E}) \leftarrow$ PARTITIONFOUNDERALLELES($\mathcal{G}(m)$)
3. **for** each $v \in \mathbf{v}$
4. **do** $\mathcal{P}(v|\mathcal{G}(m)) \leftarrow$ CALCULATEPROBABILITY($v, (\mathcal{A},\mathcal{U},\mathcal{E}), \mathcal{G}(m)$)
5. $\mathcal{P}(\mathbf{v}_m|\mathcal{G}(m)) \leftarrow$ UPDATEPROBABILITY($\mathbf{v}_m, \mathbf{v}_{m-1}, \mathbf{v}_{m+1}$)

Where $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ denotes the sets of assignment of alleles to the founder loci, $\mathbf{v_M}$ is the set of possible inheritance vectors of the pedigree at all markers.

For each marker the LANDER-GREEN Algorithm takes as input a pedigree, the set of all the possible inheritance vectors, and the genotype information given for this marker. It then calculates the probability of all founder allele assignments, where the probability of inheritance vectors leading to incompatible founder allele assignments is zero. In LANDER-GREEN's approach we talk about the loci as the founder variables, that are given alleles as values. The compatible founder allele assignments are found by applying each possible inheritance vector to the pedigree and for each genotyped non-founder assign the appropriate allele to the inherited founder loci. The inheritance vectors are then rated by probability. Finally when linkage analysis have been applied to each marker, the probability of each inheritance vector for a marker is updated with respect to the set of inheritance vectors at the neighboring markers.

For each marker the LANDER-GREEN Algorithm checks each possible inheritance vector. The number of possible inheritance vectors is 2^{2n} where n is the number of non-founders in the pedigree. This means that the algorithm is exponential in the number of persons in the pedigree. However for each additional marker the algorithm is only called once, so it is linear in the number of markers.[1]

The founder loci are divided into three sets in the LANDER-GREEN Algorithm $(\mathcal{A}, \mathcal{U}, \mathcal{E})$, where:

$\mathcal{A}$ is the set of unambiguously assigned loci, corresponding to loci for which the allele is known; ex $f_1 = \alpha_1$;

$\mathcal{U}$ is the set of free loci, corresponding to loci with no given constraints. This means that either the inheritance vector is not pointing to the loci, or that no genotyped non-founders have inherited the loci; and

$\mathcal{E}$ is the set of ambiguously assigned loci, corresponding to loci that can be assigned one of two different alleles; ex. $f_1 = \alpha_1 \vee f_1 = \alpha_2$. The loci in $\mathcal{E}$ will always be connected in components of two or more loci, where the edge between two loci means that an allele can be inherited from one loci or the other; ex. $(f_1 = \alpha_1 \wedge f_2 = \alpha_2) \vee (f_1 = \alpha_2 \wedge f_2 = \alpha_1)$, this we write $f_1 \overset{\alpha_1}{\underset{\alpha_2}{\longleftrightarrow}} f_2$. There is no maximum length of the component "chains" in $\mathcal{E}$, but if *one* locus of the chain is moved to $\mathcal{A}$, the entire chain can be unambiguously assigned alleles and moved to $\mathcal{A}$.

[1] Remember that in the Elston-Stewart approach described previously, the complexity was flipped: exponential in markers and linear in the number of individuals.

Algorithm PARTITIONFOUNDERALLELES($\mathcal{G}(m)$)
1. **for** each $n \in \mathbf{F}$
2. **if** $n \in \mathcal{L}$
3. $\alpha_1, \alpha_2 \leftarrow$ alleles(n)
4. $\mathcal{A} \leftarrow f_1 = \alpha_1, f_2 = \alpha_1$
5. **else** $\mathcal{U} \leftarrow f_1, f_2$
6. **return** $(\mathcal{A},\mathcal{U},\mathcal{E})$

PARTITIONFOUNDERALLELES is the preprocessing step of assigning alleles to founders, where $\mathcal{L}$ denotes the set of genotyped individuals in the pedigree, α_1, α_2 are the allele variables, $alleles(n)$ is the observed genotype information on individual n, and f_1, f_2 are the founder loci variables inherited by individual n.

For each inheritance vector the probability given genotype information on the non-founders is calculated in Algorithm CALCULATEPROBABILITY. This calculation is based on the placement of founder loci into the three aforementioned sets. If the inheritance vector is incompatible with the genotype information given, the probability of said vector is zero, i.e. impossible.

Algorithm CALCULATEPROBABILITY($v, (\mathcal{A},\mathcal{U},\mathcal{E}), \mathcal{G}(m)$)
1. $(\mathcal{A}',\mathcal{U}',\mathcal{E}') \leftarrow$ COMPARENONFOUNDERS($v, (\mathcal{A},\mathcal{U},\mathcal{E}), \mathcal{G}(m)$)
2. **if** $(\mathcal{A}',\mathcal{U}',\mathcal{E}') = \emptyset$
3. $\mathcal{P}(v|\mathcal{G}(m)) = 0$
4. **else** $\mathcal{P}(v|\mathcal{G}(m)) = \prod_{c \in C} \sum_{s \in x} \mathcal{P}(s)$
5. **return** $\mathcal{P}(v|\mathcal{G}(m))$

Where C is the set of components of the graph, s is a solution to a component, $\mathcal{P}(s)$ is the product of frequency with which the alleles assigned to the founder loci occur in the population.

4.3.1 Probability Calculation in Lander-Green

When calculating the probability of the founder allele assignment, we think of the founder loci as being members of a *component* (c), for which we calculate the probability. The probability of the inheritance vector is then calculated on the basis of the combination and solution of these components. A *solution* to a component is a way of compatibly assigning alleles to the component with respect to the constraints given. The components created by the founder loci in the sets $\mathcal{U}$ and $\mathcal{A}$ are all components of one founder variable f_i. A component from the set $\mathcal{E}$ is a chain as described previously: a chain of founder variables which can be assigned one of two alleles. The probability of a component is then calculated as follows, where $\pi(f_n = \alpha_i)$

denotes the allele frequency of ($\pi(\alpha_i)$) indicating that locus f_i have been assigned the value α_i.

$\mathcal{A}$ The probability of an unambiguously assigned locus is given by the population frequency of the allele, assigned to that locus. This assures that inheritance vectors which assigns rare alleles to many founders are rated by a lower probability than inheritance vectors with less assignments of rare alleles. The probability of a unambiguously assigned locus f_a is then given by $\pi(f_a = \alpha)$.

$\mathcal{U}$ A locus in this set can be assigned any allele. The probability of a given loci is the sum of the population frequencies of the possible alleles at said loci. The total probability of each free locus is therefore

$$\sum_{i=1}^{k} \pi(f_u = \alpha_i) = 1$$

where $\{1, 2, \ldots, k\}$ are all the alleles possible for the entire population, and f_u is a free loci.

$\mathcal{E}$ A solution to a chain is given by assigning the possible alleles in to the loci of the chain in the two possible configurations, and summing the probabilities of the two possible solutions. Ex. for the component $(f_1 \overset{\alpha_1}{\underset{\alpha_2}{\longleftrightarrow}} f_2 \overset{\alpha_1}{\underset{\alpha_2}{\longleftrightarrow}} f_3)$, the probability is given by

$$\pi(f_1 = \alpha_1)\pi(f_2 = \alpha_2)\pi(f_3 = \alpha_1) + \pi(f_1 = \alpha_2)\pi(f_2 = \alpha_1)\pi(f_3 = \alpha_2)$$

4.3.2 Multi Point in Lander-Green

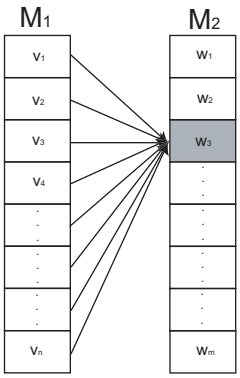


Figure 4.5: The calculation of the left-conditioned probability of one inheritance vector given the inheritance vectors at the marker to the left. Where the m in v_m is given by $m = 2^{2n}$ and n is the number of non-founders.

When the inheritance vectors at each marker has been given a probability distribution, this is updated with respect to the neighboring markers, see Figure 4.5. This means that if we have observed the exact inheritance vector at M_i to be v_3 (that is $\mathcal{P}(\mathbf{v}_i = v_3|\mathbf{G}_i) = 1$), then the probability of all inheritance vectors at M_{i+1} *close to* v_3 get increased probability, whereas inheritance vectors far from v_3 get decreased probability. When describing the distance between two inheritance vectors, the binary encoding of inheritance vectors comes in handy, because the difference[2] between two inheritance vectors can be expressed as the Hamming distance between the two. The Hamming distance is the minimum number of bits that must be changed in order to convert one bit string into another.

More precisely, the contribution from any state v_j of M_i is given by the probability of the v_j conditioned on the genotype information at that locus, $\mathcal{P}(v_j|\mathcal{G}_i)$, times the transition probability $\mathcal{P}(w_k|v_j)$. The transition probability is given by the Hamming distance between the two inheritance vectors. The probability of d recombinations occurring between two markers with an inheritance vector of length m is given by equation 4.3.

$$\mathcal{P}(w_k|v_j) = \theta_i^{d_j} \cdot (1 - \theta_i^{m-d_j}) \tag{4.3}$$

where w_k is an inheritance vector at marker M_{i+1}, v_j a vector at marker M_i, d_j is the Hamming distance between (w_k, v_j) and θ_i is the recombination fraction between the two markers.

We then sum over the contribution to w_k of every vector v_j, which corresponds to marginalizing out $\mathbf{v}_i$ (the set of inheritance vectors at M_i). The sum is then multiplied with the conditional probability of w_k given $\mathcal{G}_{i+1}$. This product is proportional to $\mathcal{P}(w_k|\mathcal{G}_i, \mathcal{G}_{i+1})$.

We compute this product for every inheritance vector at marker M_{i+1} to get the full probability distribution, $\mathcal{P}(\mathbf{v}_{i+1}|\mathcal{G}_i, \mathcal{G}_{i+1})$[3], see equation 4.4.

$$\mathcal{P}_{i+1}^L = \mathcal{P}_i^L \cdot \sum_{\mathbf{v}_i} \mathcal{P}(\mathbf{v}_i|\mathcal{G}_i)\mathcal{P}(\mathbf{v}_{i+1}|\mathbf{v}_i), 1 \leq i < m \tag{4.4}$$

The updated probability distribution of marker M_{i+1} can then be used to calculate the left-conditioned probability at marker $i + 2$ and so forth. The right-conditioned probability is calculated in a similar fashion. Using this procedure we can compute $\mathcal{P}(\mathbf{v}_i|\mathcal{G}_{all})$ for any $\mathbf{v}_i$, see equation 4.5. These values can then be used in the scoring functions, i.e. LOD score, to determine linkage between markers and traits.

$$\mathcal{P}(\mathbf{v}_i|\mathcal{G}_{all}) \propto P_i^L \cdot \mathcal{P}(\mathbf{v}_i|\mathcal{G}_i) \cdot \mathcal{P}_i^R \tag{4.5}$$

[2]Caused by recombination.

[3]Given the assumption that the result is normalized

4.3.3 Lander-Green Example

For the example in Section 4.1 the LANDER-GREEN takes in the pedigree, the genotype information given and a set of possible inheritance vectors of size 2^8. To clarify the run of the algorithm we run two vectors:
$v_1 = pppppppp$ and $v_2 = ppmppmmp$ on marker M_1. In Figure 4.6 v_2 is shown.

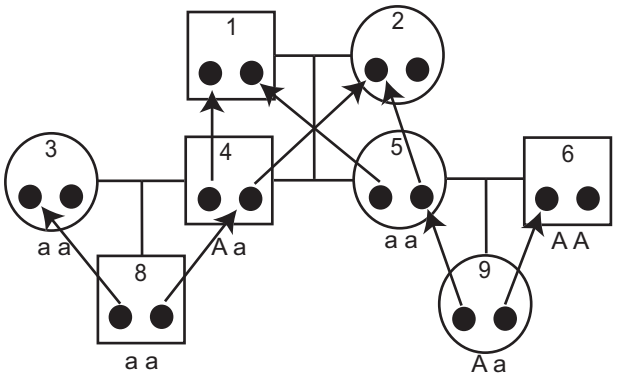


Figure 4.6: The pedigree of the example, with inheritance vector v_2 applied.

First we calculate probability of v_1 (everyone was given the paternal allele from their parents):

Step 1: PARTITIONFOUNDERALLELES. The genotyped founders are assigned alleles giving the $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ sets:

- $\mathcal{A} = \{f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
- $\mathcal{U} = \{f_1, f_2, f_3, f_4\}$
- $\mathcal{E} = \emptyset$

Step 2: CALCULATEPROBABILITY. To calculate the probability we compare the genotype information given on the non-founders, one individual at a time. The founder loci are assigned allele values and divided into sets of loci with different constraints.

1. Individual 4 is given genotype information $\mathcal{G}_4 = \{Aa\}$, for inheritance vector pp, which gives individual 4 the founder variables (f_1, f_3). The $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ sets are now:
   - $\mathcal{A} = \{f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
   - $\mathcal{U} = \{f_2, f_4\}$
   - $\mathcal{E} = \{f_1 \overset{a}{\underset{A}{\longleftrightarrow}} f_3\}$

2. Individual 5 is given genotype information $\mathcal{G}_5 = \{aa\}$, for inheritance vector pp, which gives individual 5 the founder variables (f_1, f_3), but this would mean that both f_1 and f_3 should be assigned a, and this cannot be. The inheritance vector is incompatible and
   - $(\mathcal{A}, \mathcal{U}, \mathcal{E}) = \emptyset$ and the probability $\mathcal{P}(\mathbf{v}_1 = v_1) = 0$.

Now the probability of v_2:

Step 1: PARTITIONFOUNDERALLELES. Again the genotyped founders are assigned alleles, and $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ are:

- $\mathcal{A} = \{f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
- $\mathcal{U} = \{f_1, f_2, f_3, f_4\}$
- $\mathcal{E} = \emptyset$

Step 2: CALCULATEPROBABILITY.

1. Individual 4 is given genotype information $\mathcal{G}_4 = \{Aa\}$, for inheritance vector pp
   - $\mathcal{A} = \{f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
   - $\mathcal{U} = \{f_2, f_4\}$
   - $\mathcal{E} = \{f_1 \xleftrightarrow[A]{a} f_3\}$
2. Individual 5 is given genotype information $\mathcal{G}_5 = \{aa\}$, for inheritance vector mp.
   - $\mathcal{A} = \{f_1 = A, f_2 = a, f_3 = a, f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
   - $\mathcal{U} = \{f_4\}$
   - $\mathcal{E} = \emptyset$ etc. the founder set will not change through adding more information from non-founders.
3. The probability calculation is given by
   $\pi(f_1 = A)\pi(f_2 = a)\pi(f_3 = a)\pi(f_5 = a)\pi(f_6 = a) \cdot$
   $\pi(f_{11} = A)\pi(f_{12} = A)$
   $= (0.25)^3(0.75)^4 = 0.004943$

Actually the inheritance vectors end up dividing into three sets of equiprobable vectors.

1. $\mathbf{v}_{invalid} = 0$,
2. $\mathbf{v}_{low} = 0.00308$ and
3. $\mathbf{v}_{high} = 0.004943$

4.4 The Fast Tree Traversal Algorithm

Allegro is the software package for genetic analysis developed by DeCode. The algorithm used for linkage analysis is called FASTTREETRAVERSAL. It traverses through the pedigree, and builds a BDD structure (see Appendix C) of possible inheritance vectors. The algorithm is a modification of the approach developed by Lander-Green. Our knowledge of the algorithm is based on [16].

Algorithm FASTTREETRAVERSAL($P, \mathcal{G}(\mathbf{M})$)
1. **for** each $m \in \mathbf{M}$
2. **do for** each $n \in \mathbf{F}$
3. **do** PARTIONTIONFOUNDERALLELES($n, (\mathcal{U}, \mathcal{E}, \mathcal{A})$)
4. **for** each $n \in \mathbf{N}$
5. **do for** each $(x, y) \in \{(p, p), (p, m), (m, p), (m, m)\}$
6. **do** $v' \leftarrow \{v \in \mathbf{v} | v(n, p) = x \wedge v(n, m) = y\}$
7. $(\mathcal{A}', \mathcal{U}', \mathcal{E}') \leftarrow (\mathcal{A}, \mathcal{U}, \mathcal{E})$
8. **if** $n \in \mathcal{L}$
9. $(\mathcal{A}', \mathcal{U}', \mathcal{E}') \leftarrow$
10. PARTITIONFORNONFOUNDERS($v', n, (\mathcal{A}', \mathcal{U}', \mathcal{E}')$)
11. **if** $(\mathcal{A}', \mathcal{U}', \mathcal{E}') \neq$ **incompatible**
12. FASTTREETRAVERSAL($v', (\mathcal{A}', \mathcal{U}', \mathcal{E}')$)
13. $\mathcal{P}(\mathbf{v}_m | \mathcal{G}(m)) \leftarrow$ UPDATEPROBABILITY($\mathbf{v}_m, \mathbf{v}_{m-1}, \mathbf{v}_{m+1}$)

Where x is the locus inherited from the father of n, and y is the locus inherited from the mother of n, and (x, y) indicates the two bits of the inheritance vector which depicts the inheritance pattern of n. $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ are the sets of founder loci, as described in the previous section.

The overall structure of the FASTTREETRAVERSAL Algorithm is identical to the LANDER-GREEN Algorithm, however instead of checking all the possible inheritance vectors of the pedigree, it builds only the *compatible* inheritance vectors.

The algorithm traverses through the pedigree, and for each non-founder n the genotype information given for n ($\mathcal{G}_n$), is checked for compatibility with the current division of founder loci into the three $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ sets. In this fashion a tree-structure of inheritance vectors are build one non-founder at a time, terminating a path if it shows incompatibility. In reality many inheritance vectors at each marker are incompatible, and therefore aborting these as soon as incompatibility is discovered saves a lot of time, in comparison to the Algorithm LANDER-GREEN[4]. However in the worst case scenario the FASTTREETRAVERSAL Algorithm will still build all possible inheritance

[4]This is based on the average case.

vectors of the markers, and the complexity is therefore still exponential in the number of non-founders of the pedigree. Markers where that many of the possible inheritance vectors are compatible are called *highly uninformative* markers. A highly uninformative marker indicate that a low percentage of individuals (or none) have been genotyped for this marker or most individuals have the same genotype. At highly *informative* markers most inheritance vectors are incompatible.

The first step of the FASTTREETRAVERSAL Algorithm is (as in LANDERGREEN) to divide the founder loci into the three sets, based on the genotype information given at founder level.

Then for each genotyped non-founder n of the pedigree the inherited founder loci f_1, f_2 is compared with the given genotype information for n, and either the constraints on the sets are tightened[5], stays the same or found incompatible, and the path is discarded. The set constraints are updated through a call to PartitionForNonFounders. PartitionForNonFounders handles the new genotype information, with respect to six possible configurations of the inherited founder loci:

1. Both founder loci are in $\mathcal{U}$. See Figure 4.7.
   (a) If n is homozygous ($\alpha_1 = \alpha_2$), then $f_1 = \alpha_1$ and $f_2 = \alpha_1$ are moved to $\mathcal{A}$.
   (b) If n is heterozygous the component $f_1 \overset{\alpha_1}{\underset{\alpha_2}{\longleftrightarrow}} f_2$ is moved to $\mathcal{E}$.

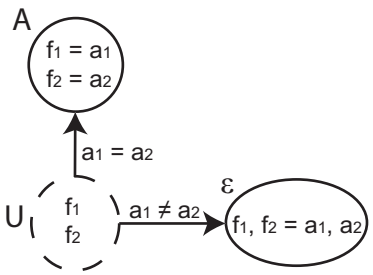


Figure 4.7: Initially both the founder loci are in $\mathcal{U}$. If the genotype of the non-founder n is homozygous, then the two loci are assigned alleles, and moved to $\mathcal{A}$. If n is heterozygous they are combined in a component and assigned to $\mathcal{E}$.

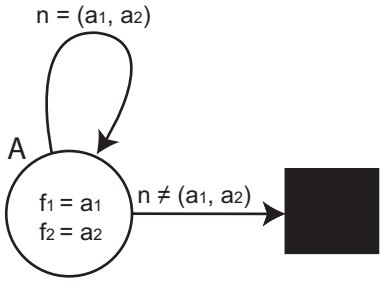


Figure 4.8: Initially both the founder loci are in $\mathcal{A}$. If the genotype of the non-founder n is not compatible with the possible alleles assigned to the two loci, the path is terminated.

2. Both founder loci are in $\mathcal{A}$. See Figure 4.8.
   (a) The genotype of n is compared to the assigned alleles of the founder loci. If they are incompatible the path is terminated.

[5]Moving founder loci from $\mathcal{U}$ to $\mathcal{E}$, or $\mathcal{E}$ to $\mathcal{A}$, tightens the constraints.

3. Both founder loci are in $\mathcal{E}$. See Figure 4.9.

   (a) If n is homozygous ($\alpha_1 = \alpha_2$), and α_1 is a possible value for both loci, then $f_1 = \alpha_1$ and $f_2 = \alpha_1$ moved to $\mathcal{A}$, as are the loci in the same components as f_1 and f_2, and so forth.

   (b) If n is heterozygous **and** α_1 and α_2 are possible values of both founder loci, the algorithm do a split and generates two paths, one with $f_1 = \alpha_1$, $f_2 = \alpha_2$ moved to $\mathcal{A}$ and one with $f_1 = \alpha_2$, $f_2 = \alpha_1$ moved to $\mathcal{A}$. This means that there are two versions of the same inheritance vector, but resulting in two different sets of constraints on the $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ sets. Later in this chapter we will go into the split operation in more detail.

   (c) If α_1 is a possible assignment of f_1 but not f_2, and α_2 is a possible assignment of f_2, the $f_1 = \alpha_1$ and $f_2 = \alpha_2$ are moved to $\mathcal{A}$.

   (d) If one of the alleles does not correspond to any legal assignment of any of the two loci, the path is terminated.

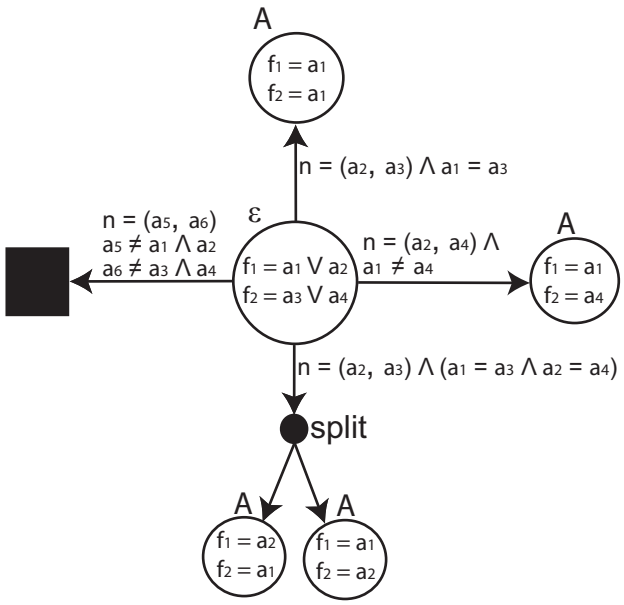


Figure 4.9: Initially both the founder loci are in $\mathcal{E}$. If the genotype of the non-founder n is not compatible with any one of the loci, the path is terminated. Else the loci are moved to $\mathcal{A}$.

4. One locus is in $\mathcal{A}$ and the other is in $\mathcal{U}$. See Figure 4.10.

   (a) If one allele of n corresponds to the allele of the locus in $\mathcal{A}$, then the locus in $\mathcal{U}$ can be assigned a specific allele, and moved to $\mathcal{A}$. Ex. take $n = \alpha_1, \alpha_2$; $(f_1 = \alpha_1) \in \mathcal{A}$ and $f_2 \in \mathcal{U}$. Then $f_2 = \alpha_2$ and is moved to $\mathcal{A}$.

(b) If no allele of n corresponds to the value of $f_1 \in \mathcal{A}$, the path is terminated.

5. One locus is in $\mathcal{A}$ and one is in $\mathcal{E}$. See Figure 4.11.

   (a) If $(f_1 = \alpha_1) \in \mathcal{A}$ **and** the α_2 corresponds to one of the possible alleles of $f_2 \in \mathcal{E}$, then $f_2 = \alpha_2$ is moved to $\mathcal{A}$.

   (b) The path is terminated if none of the alleles of n correspond to the possible alleles of either $f_1 \in \mathcal{A}$ or $f_2 \in \mathcal{E}$.

6. One locus is in $\mathcal{E}$ and one is in $\mathcal{U}$. See Figure 4.12.

   (a) If both alleles of n are possible values of $f_1 \in \mathcal{E}$ a split is made, as in 3.

   (b) If only one allele α_1 is a possible assignment of $f_1 \in \mathcal{E}$, then $f_1 = \alpha_1$ and $f_2 = \alpha_2$ are moved to $\mathcal{A}$ together with a all the other loci of the component of f_1, as in 3a.

   (c) If there are no possible allele assignment of f_1 the path is terminated.

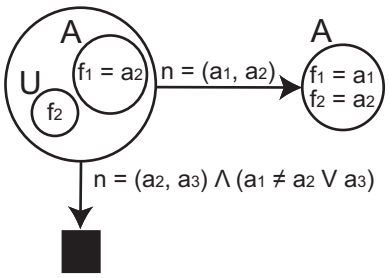


Figure 4.10: Initially one founder locus is in $\mathcal{A}$ and one is in $\mathcal{U}$. If the genotype of the non-founder n is incompatible with the assigned allele of the locus in $\mathcal{A}$ the path is terminated. Else the locus in $\mathcal{U}$ is assigned an allele and moved to $\mathcal{A}$.

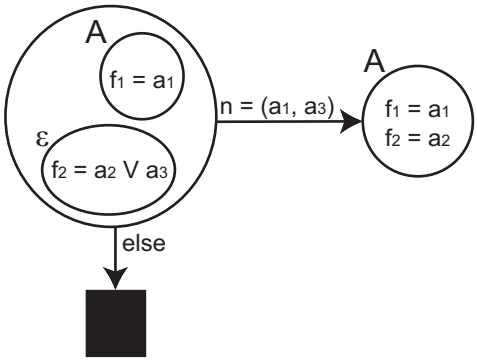


Figure 4.11: Initially one founder locus is in $\mathcal{A}$ and one is in $\mathcal{E}$. If the genotype of n is incompatible with either the allele of the locus in $\mathcal{A}$ or both the possible alleles of the locus in $\mathcal{E}$, the inheritance vector is terminated. Else both alleles are now unambiguously assigned alleles, as are all loci in the affected component in $\mathcal{E}$.

The calculation of the probability distributions for the inheritance vectors and the multi point update of the probability distributions are calculated as in Section 4.3.

In reality it turns out that not only are there many incompatible inheritance vectors that go straight to a terminal node of value $= 0$, the compatible in-

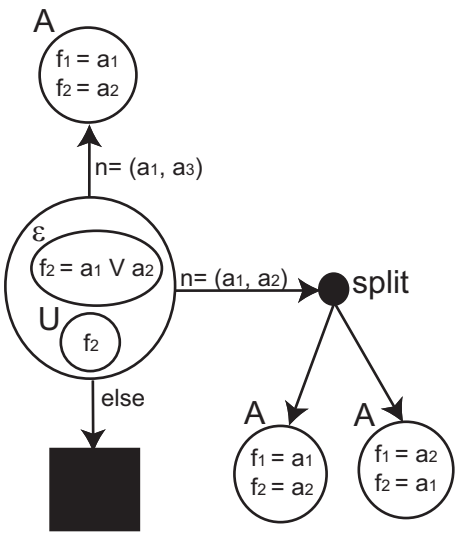


Figure 4.12: Initially one locus is in $\mathcal{E}$ and one is in $\mathcal{U}$. If the genotype of n is incompatible with both possible alleles of the locus in $\mathcal{E}$ the inheritance vector is terminated, else both founder loci are unambiguously assigned alleles, as are all loci in the affected component in $\mathcal{E}$.

heritance vectors also come together in sets of equiprobable vectors[6]. This is an intuitive feature, which is based on the fact that the probability of an inheritance vector is calculated as the probability of the compatible founder allele assignment.

DD data structures have proven successful in exploiting symmetries for compact symbolic representation of large state spaces, as for instance identifying inheritance vectors with probabilities. 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. See Appendix C for an introduction to BDDs and MTBDDs.

FASTTREETRAVERSAL takes advantage of the symmetries of the inheritance vectors by reducing the inheritance vector trees to Multi Terminal BDDs (MTBDD). FASTTREETRAVERSAL actually dynamically builds the MTBDD version of the inheritance vector tree, and reuses thereby identical subtrees. However it is conceptually easier to think of the structure being build as a tree and then merged bottom-up.

[6] As in the example in Section 4.3.3.

4.4.1 Split Operation

In the FASTTREETRAVERSAL Algorithm, they in two cases incorporate a *split* operation.

- Both founder variables are in the set $\mathcal{E}$, the genotyped individual n is heterozygous and α_1 and α_2 are possible values of both founder loci.
- One founder variable is in $\mathcal{E}$, the other is in $\mathcal{U}$, and both alleles of individual n are possible values of $f_1 \in \mathcal{E}$.

The split operation creates two copies of the same inheritance vector. For each copy of the inheritance vector, the implicated founder variables are all moved to the set $\mathcal{A}$. In one copy $f_1 = \alpha_1, f_2 = \alpha_2$ and so forth. In the other $f_1 = \alpha_2, f_2 = \alpha_1$. This means that for every node in the set of inheritance vector a list of possible $\mathcal{A}, \mathcal{U}, \mathcal{E}$) sets are maintained, **and** that the maximum length of a component $c \in \mathcal{E}$ is $|c| = 2$.

4.4.2 Founder Reduction

The FASTTREETRAVERSAL Algorithm also employs another way of reducing the number of inheritance vectors, known as *founder reduction*. The intuitive idea is that since we do not have any information on the parents or sibling relations of the founders, we cannot deduce any knowledge on the phase of the founder alleles. This means that all the inheritance vectors which describe the same founder allele as being inherited from the founders mother or from the founders father, are equiprobable and cannot be differentiated. Replacing all different inheritance vectors which stem from this phenomenon with a single vector, reduces the amount of inheritance vectors under investigation. This is not really a reduction of possible inheritance vectors, but an exploitation of the symmetric nature of the vectors.

4.4.3 Fast Tree Traversal Example

When building the set of possible inheritance vectors for marker M_1 on the pedigree in the example given in Section 4.1 the FASTTREETRAVERSAL runs as follows :

Step 1: PARTITIONFOUNDERALLELES. The genotyped founders are assigned alleles.

- $\mathcal{A} = \{f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
- $\mathcal{U} = \{f_1, f_2, f_3, f_4\}$
- $\mathcal{E} = \emptyset$

Step 2: PartitionForNonFounders. By assigning and comparing the genotype information given on the non-founders, one individual at a time, the founder loci are assigned allele values and divided into sets of loci with different constraints.

1. Individual 4 is given genotype information $\mathcal{G}_4 = \{Aa\}$, and starts by building inheritance vector pp
   - $\mathcal{A} = \{f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
   - $\mathcal{U} = \{f_2, f_4\}$
   - $\mathcal{E} = \{f_1 \xleftrightarrow[A]{a} f_3\}$
2. Individual 5 is given genotype information $\mathcal{G}_5 = \{aa\}$, and the algorithm starts by adding pp to the inheritance vector. This however is incompatible, and the path lead to terminal node 0.
3. Then the algorithm tries to add pm to the inheritance vector. This is also incompatible, and the path is terminated, and lead to 0.
4. When it tries to add mp the path is valid and further constraints are made to the founder sets:
   - $\mathcal{A} = \{f_1 = Af_2 = a, f_3 = a, f_5 = a, f_6 = a, f_{11} = A, f_{12} = A\}$
   - $\mathcal{U} = \{f_4\}$
   - $\mathcal{E} = \emptyset$ etc.

In LANDER-GREEN we saw that the inheritance vectors of the example are divided into three sets of equiprobable inheritance vectors, which is why building the vectors as a MTBDD reduces the space needed for the inheritance vectors. Taking a closer look at the inheritance vectors we see that all the non-founder of the 3rd generation does not contribute to any probability information, see Figure 4.13. However the compatible inheritance vectors are actually saved in the MTBDD.

4.5 Superlink

Superlink [9] was developed at Haifa university in Israel by Dan Geiger et al. It uses Bayesian networks to perform linkage analysis.

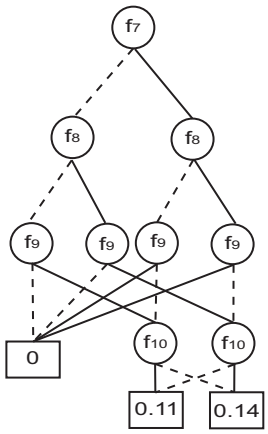


Figure 4.13: The MTBDD of inheritance vectors build for marker M_1 of the example, when doing linkage analysis using the LANDER-GREEN Algorithm. The MTBDD has been greatly reduced, because no new constraints are added to the three sets, by any of the non-founders of the 3rd generation. In reality however the inheritance vectors for all the non-founders are saved in the MTBDD, we only reduced the size for simplicity.

Superlink can perform linkage analysis on a broad variation of pedigrees. From small pedigrees with many loci to big pedigrees which fewer loci. When they have a small pedigree they use an approach like the one used in the LANDER-GREEN Algorithm, Section 4.3, peeling one locus at a time. If it is a big pedigree it uses a approach more like the Elston-Stewart approach in Section 4.2, peeling one nuclear family at a time. Generally Superlink utilizes a mix of both.

Superlink uses a *segregation network* to represent the inheritance pattern in the pedigree [28]. For each individual i at a marker we have two nodes which represent the maternally and paternally inherited alleles respectively. The underlying random variables can assume any of the allele types at that marker. Additional nodes representing the meiosis indicators are added as parents to each allele node. These are binary nodes assuming the value 1 to denote that a copy of the paternal allele of the corresponding parent was inherited and 0 to indicate inheritance of the maternal allele.

Although the genotypes may be observable in many cases, in some situations only phenotype information is available. This is modelled in a phenotype node, which is a child of the two allele nodes for each individual. The local Markov property of the Bayesian network augmented with phenotypic information is ensured by the phenotype Y_i of any individual being conditionally independent of other variables in the network, given the genotype $\mathcal{G}_i$ of said individual.

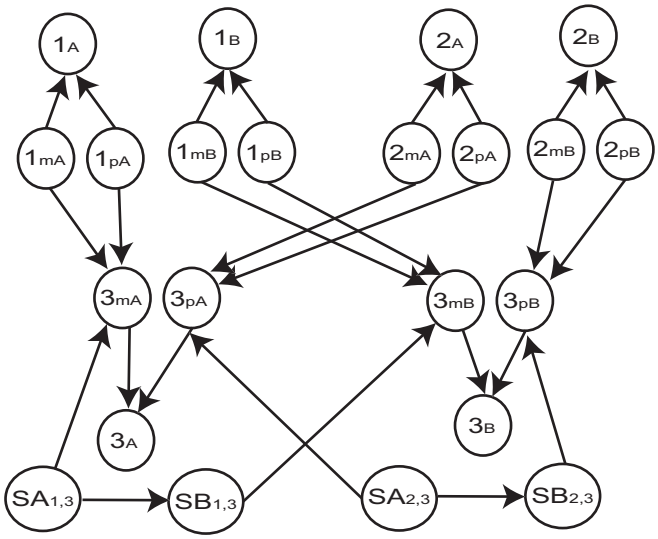


Figure 4.14: An example network for Superlink given a pedigree of three individuals, two founders and one non-founder. The node labels are read as follows: 1_{mA} is the maternal genotype node, individual 1, marker A. 1_A is the phenotype node for individual 1. $SA_{1,3}$ is the selector node between individuals 1 and 3, at marker A.

To sum up: the variable types in Superlink are:

- **Genetic loci:** For each individual i on each locus j two random variable $\mathcal{G}_{i,jp}$ and $\mathcal{G}_{i,jm}$ are defined, where $\mathcal{G}_{i,jp}$ defines the paternal allele for individual i on locus j, and $\mathcal{G}_{i,jm}$ defines the maternal allele.
- **Phenotypes:** For each individual i and each locus j a variable $Ph_{i,j}$ is defined to denote the phenotype of individual i on locus j.
- **Selector variable:** This variable describes the inheritance patterns of the pedigree, corresponding to the vectors of the LANDER-GREEN Algorithm, see Section 4.3. The variables $\mathcal{S}_{i,jp}$ and $\mathcal{S}_{i,jm}$ are used to describe the probabilities of inheriting the parental maternal and paternal alleles. These variables are binary, and are interpreted as follows: if a denotes i's father, then:

$$\mathcal{G}_{i,jp} = \begin{cases} \mathcal{G}_{a,jp} & \text{if } \mathcal{S}_{i,jp} = 0 \\ \mathcal{G}_{a,jm} & \text{if } \mathcal{S}_{i,jp} = 1 \end{cases}$$

  $\mathcal{G}_{i,jm}$ is defined in similar fashion.

The probability tables of the Bayesian network is of the following form:

- **Transmission models:** $\mathcal{P}(\mathcal{G}_{i,jp}|\mathcal{G}_{a,jp}, \mathcal{G}_{a,jm}, \mathcal{S}_{i,jp})$, $\mathcal{P}(\mathcal{G}_{i,jm}|\mathcal{G}_{b,jp}, \mathcal{G}_{b,jm}, \mathcal{S}_{i,jm})$, where a and b are i's parents. I.e. the probability of the genotype of i given i's parents.

- **Penetrance models:** $\mathcal{P}(Ph_{i,j}|\mathcal{G}_{i,jp}, \mathcal{G}_{i,jm})$. The probability of phenotype of i given the genotype of i.
- **Recombination models:** $\mathcal{P}(\mathcal{S}_{i,jp}|\mathcal{S}_{i,j-1p}, \theta_{j-1})$ and $\mathcal{P}(\mathcal{S}_{i,jm}|\mathcal{S}_{i,j-1m}, \theta_{j-1})$, where θ_{j-1} is the recombination frequency between locus j and $j-1$. I.e. the probability of inheriting parental alleles given the inheritance pattern at the neighboring locus, and the recombination fraction between the two loci. At locus 1 there is an equal probability of inheriting the one or the other parental allele;
$\mathcal{P}(\mathcal{S}_{i,1p}) = \mathcal{P}(\mathcal{S}_{i,1m}) = (0.5, 0.5)$.
- **General population allele probabilities:** $\mathcal{P}(\mathcal{G}_{i,jp})$, $\mathcal{P}(\mathcal{G}_{i,jm})$, where i is a founder. The probability of the founder alleles are given by the allele population frequencies.

There are four modules in the Superlink Algorithm: UPDATEPEDIGREE, VARIABLETRIMMING, MERGEVARIABLES and MARGINALIZATION.

At first in UPDATEPEDIGREE the genotype information is propagated through the pedigree. This is done into two phases. First the information of the network is updated, such that nodes without information are updated according to the evidence entered at other nodes of the network. For instance if the genetic evidence is entered for a parent, constraints can be put on the possible genotypes of the child, and so forth. The second phase is reduction of the probability tables, where rows and columns consisting only of invalid values are removed. An example, if $\mathcal{P}(x, y, z) = 0$ for all values of y and z of $\mathbf{Y}, \mathbf{Z}$ then the value x is not valid for the variable $\mathbf{X}$.

Algorithm UPDATEPEDIGREE($\mathbf{P}, \mathcal{G}$)
1. **for** each $n \in P$
2. **do if** informationOnParents $\wedge$ noInformationOnChildren
3. updateChildrenInformaion();
4. **if** noInformationOnParents $\wedge$ informationOnChildren
5. updateParentsInformation();
6. **if** InvalidValue
7. removeInvalidValue();

When the information has been propagated through the segregational network the barren phenotype variables can be removed, see Appendix B. These are either phenotype nodes without evidence, or phenotype nodes for which both the genotype nodes have been given evidence. These variables do not add any information to the calculations of probability distribution of the possible inheritance patterns.

In VARIABLETRIMMING the leaves of the network are checked for evidence. The leaves without evidence are removed. This is a recursive step, meaning

that when a leaf is deleted, the new leaves are also checked to see if they had been given evidence and so forth.

Algorithm VARIABLETRIMMING($\mathbf{P}, \mathcal{G}$)
1. **for** each $n \in \mathbf{P}$
2. **if** $n \in$ leaf $\wedge$ noInformationOnLeaf
3. delete leaf;
4. **if** $\mathcal{G}_n ! = \emptyset \vee Ph_n == \emptyset$
5. deletePhenotypeNode();

Then all redundant variables are merged in MERGEVARIABLES. An example of variables which are redundant are the paternal and maternal alleles at the founder level. This corresponds to founder reduction i the LANDER-GREEN Algorithm. If there is no genotype information given for the founder, it is not possible to calculate the inheritance pattern of a child of the founder so the selector variable of the child is also redundant.

Algorithm MERGEVARIABLES($\mathbf{P}, \mathcal{G}$)
1. **for** each $n \in \mathbf{P}$
2. **if** $n \in \mathbf{F} \wedge \mathcal{G}_n == \emptyset$
3. merge(founderLociInformation);
4. delete($\mathcal{S}_n$ for children);

Now Superlink projects down on each marker to find the most probable inheritance pattern, i.e. the most probable configuration of the Bayesian network. This is done either by variable elimination or conditioning, see Algorithm MARGINALIZATION. Variable elimination is a very expensive process to perform, i.e.it requires a lot of space, but it is faster than conditioning. Conditioning however is not as expensive with respect to space. Therefore variable elimination is used as much as possible. To determine, wether it is too expensive to perform variable elimination, the maximum cost of removing a given variable is calculated and compared to a predefined threshold.

The segregation graph is non-triangulated, and a greedy algorithm is used to find the best elimination order[7]. Superlink uses either a deterministic greedy algorithm or a stochastic greedy algorithm. The deterministic algorithm finds a good[8] fill-in combination for the entire graph, before it starts to eliminate variables. The stochastic algorithm calculates fill-in sizes randomly in the network and starts variable elimination if the fill-in size is below a predefined threshold [10].

[7]The elimination order resulting in the cheapest fill-ins, see Appendix B.
[8]However probably not the optimal.

Algorithm MARGINALIZATION($\mathbf{P}, \mathcal{G}$)
1. **if** $\frac{max}{v} n(v)EC(v)$ <threshold
2. **if** deterministiskTime $< C_{min}$
3. DeterministiskGreedy();
4. **else**
5. StochasticGreedy();
6. PerformVariableElimination();
7. **else**
8. Conditioning();

4.5.1 Superlink Example

Given the example pedigree in Section 4.1, Superlink builds a segregation graph as in Figure 4.15. In the beginning the graph contains 64 nodes (5 per non-founder per marker, and 3 per founder per marker), compared to Elston-Stewart's graph containing 18 nodes. However the total size of the joined probability table is (when ignoring the phenotype nodes) in Superlink 2^{32}, and in Elston-Stewart 9^{16} (about 431,440 times bigger than Superlink). The size of the joined table is given by taking the product of the number of possible values of each variable, for all the variables, which is shown in equation 4.6.

$$size = \prod_{i=1}^{k} r_i(|\alpha|) \tag{4.6}$$

where i is the number of variables in the graph and $|\alpha|$ is the number of possible values of the variable.

When evidence has been entered and the impossible values have been removed the joined table for the segregation graph is reduced to 2^4. In the example there is no phenotype information given for any of the persons, therefore all the phenotype nodes are removed in VARIABLETRIMMING.

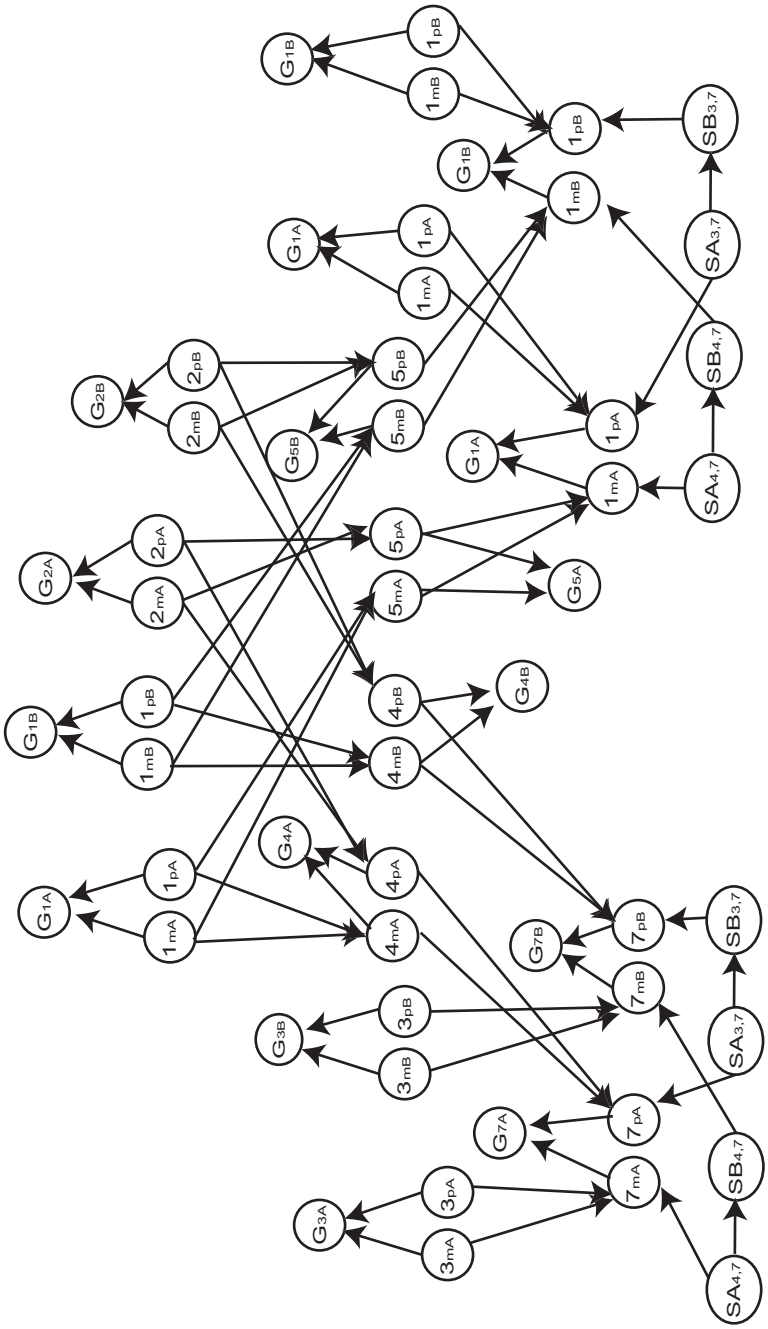


Figure 4.15: The segregation graph in Superlink for the example pedigree. The phenotype nodes have been left out, to simplify the picture. They are all barren and therefore removed anyway. The only time a phenotype node is left in the graph is when it has been given evidence, and the genotype nodes have not.

Summary

In this chapter we have looked at four linkage analysis approaches:

- the Elston-Stewart Algorithm,
- the Lander-Green Algorithm,
- The Fast Tree Traversal Algorithm, and
- the Superlink Algorithm.

The Elston-Stewart Algorithm was not developed for linkage analysis, however the architecture and ideas developed in this approach have been the basis of many genetic analysis algorithms developed at a later time.

The Elston-Stewart algorithm calculates the likelihood for one nuclear family at a time. The genotype of an individual is seen as a string of information, which can represent several markers or traits. The number of possible genotype configurations under investigation grow exponentially in the number of markers. Thereby is the complexity of the algorithm linear in the number of people in the pedigree and exponential in the number of markers.

The Lander-Green Algorithm peels one entire pedigree for a single marker at a time.

Lander-Green starts by doing single point analysis for each marker, and then proceeds to update the probability distribution of inheritance patterns at one marker with respect to the neighboring markers.

Lander-Green encodes inheritance patterns as binary vectors, where each bit denotes one inherited allele of a non-founder, and the value of each bit describes whether the inherited allele is the maternal or paternal allele of the parent. The total inheritance vector v of a pedigree is a concatenation of bit pairs for each non-founder.

The compatible founder allele assignments are found by applying each *possible* inheritance vector to the pedigree and for each genotyped non-founder assign the appropriate allele to the inherited founder loci.

Finally when linkage analysis have been applied to each marker, the probability of each inheritance vector for a marker is updated with respect to the set of inheritance vectors at the neighboring markers.

The LANDER-GREEN Algorithm is exponential in the number of persons in the pedigree and linear in the number of markers.

The Fast Tree Traversal Algorithm traverses through the pedigree, and builds a BDD structure of possible inheritance vectors.

The overall structure of the FASTTREETRAVERSAL Algorithm is identical to the LANDER-GREEN Algorithm, however instead of checking all the possible inheritance vectors of the pedigree, it builds only the *compatible* inheritance vectors. This is a great optimization compared to LANDER-GREEN, however the worst case compatibility is the same.

The Superlink Algorithm can perform linkage analysis on a broad variation of pedigrees. From small pedigrees with many loci to big pedigrees which fewer loci. For a small pedigree they use an approach like the one used in the LANDER-GREEN Algorithm. For a big pedigree it uses a approach like the Elston-Stewart approach. This diversity comes from Superlink using Bayesian networks.

5 Linkage Analysis Algorithm Design

Generally speaking the task of doing linkage analysis follow the same structure for all the algorithms. The goal is to find the set of inheritance patterns for a marker, given some genotype information on the individuals of a pedigree, and rate these inheritance patterns by the probability of the compatible founder allele assignments. Thereby there are four basic components in linkage analysis: pedigree founder genotype information $F_{\mathcal{G}}$, genotype information on the non-founders $\mathcal{G}$, phenotype information[1] Ph and finally the sets of inheritance patterns I. Figure 5.1 illustrates the dependencies between these four components, using a Bayesian Network (BN) [19]. The phenotypes of the individuals are dependent on their genotypes, which in turn are dependent on the genotypes of the founders and the inheritance patterns.

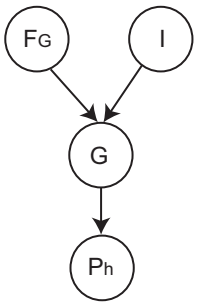


Figure 5.1: Linkage Analysis as a Bayesian Network (BN) [19]. **Note:** each node holds *all* the information for *all* individuals of the set across all markers.

[1] Phenotype information is often regarded as partial genotype information.

The probability distributions of the inheritance patterns are calculated on basis of observations given on $F_{\mathcal{G}}, \mathcal{G}$ and Ph. However in practise the probability tables at this abstraction level are impossibly large, and therefore no feasible solutions exists using only the four basic variables.

There are two major branches or types of algorithms for doing linkage analysis. These are defined by their inference flow, which either run across markers or across nuclear families[2]. The branches are named after the first algorithms created in each: the Lander-Green branch and the Elston-Stewart branch. The algorithms are described in Sections 4.3 and 4.2 respectively. When other algorithms refer to these, they are actually referring to the inference flow.

The Lander-Green Branch: In Lander-Green based approaches the inheritance patterns are first built for each marker in single point analysis. When doing multi point analysis the inheritance patterns are updated with information given on the neighboring markers, see Figure 5.2.

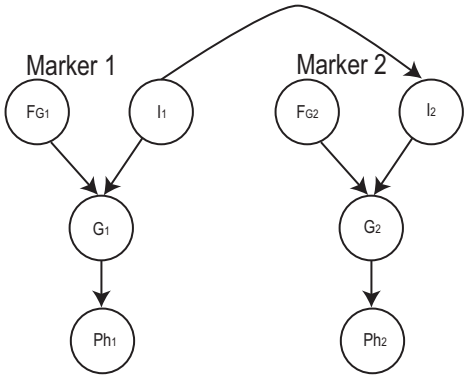


Figure 5.2: Linkage analysis is performed in a single point fashion at each marker. Multi point dependencies only exists between the inheritance patterns of neighboring markers. Each variable node holds information only for a single marker, but across sets of founders and non-founders.

The Elston-Stewart Branch: The Elston-Stewart based algorithms focus on each individual in the pedigree. The genotype dependencies are now specified between parents and children, and not as direct dependency between founders and non-founders, see Figure 5.3. In multi point linkage analysis each node contains genotype information across all the markers.

The inference flow of the algorithms is based on which order the variables are marginalize out of the graph. The marginalization order is also called

[2] A nuclear family is a set of parents and their kids.

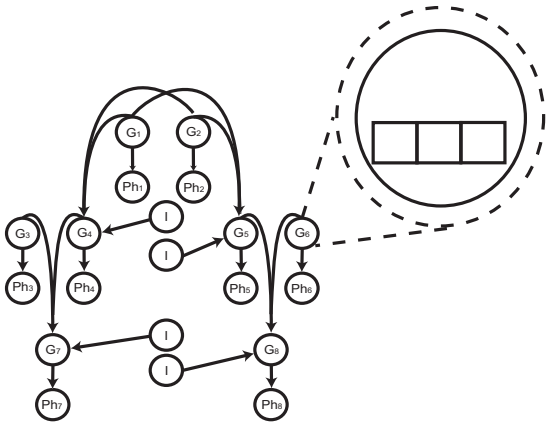


Figure 5.3: Each node holds information across all markers for a single individual. This means that linkage analysis is performed in a multi point fashion, but only for one nuclear family at a time.

the *elimination* order. The inference flow of an algorithm is illustrated using a junction tree, which is an inference tool for Bayesian networks, see Appendix B.

In single point analysis each node, in the Bayesian network for Elston-Stewart, only holds information on a single marker. This results in an inference flow across nuclear families as in Lander-Green. This is demonstrated graphically by the junction tree which is identical for the two algorithms in single point analysis. As an example see the junction tree in Figure 5.5 for the pedigree in Figure 5.4.

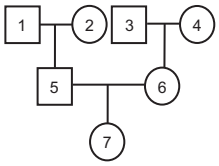


Figure 5.4: An example pedigree.

General for all the algorithms is that the nodes of the founders are triangulated, and that additional edges between the nodes of the non-founders are needed to triangulate these. The additional edges are called *fill-ins*.

The selector nodes encode the inheritance patterns, and therefor the goal of the algorithms is to calculate the probability distribution of the selector nodes. The joint distributions of the inheritance patterns are given by *pushing* the values of the rest of the graph into the selector nodes. The major difference in the inference flow of the different algorithms is seen in respect to the selector nodes.

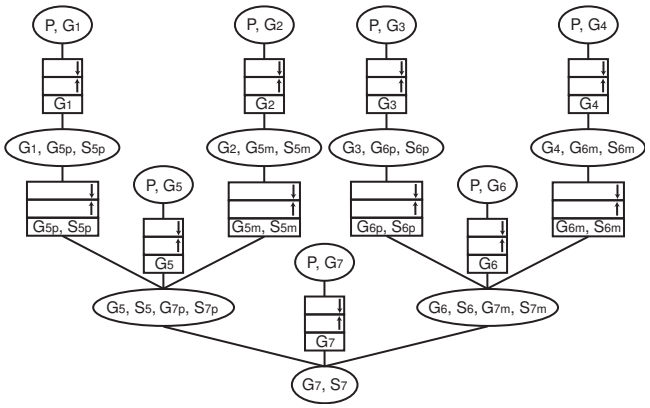


Figure 5.5: The single point junction tree given the pedigree in Figure 5.4.

In multi point analysis the choice of which type of algorithm to employ depends on the input data. Superlink incorporates both branches by building a large Bayesian network, where each node only holds information on a single individual at a single marker, see Figure 5.6. Actually Superlink divides the information even further such that for each individual there are 5 nodes, see Section 4.5, but with respect to the direction of the inference flow this is un-important. The large Bayesian network built by Superlink lets the inference flow run both across markers and across nuclear families.

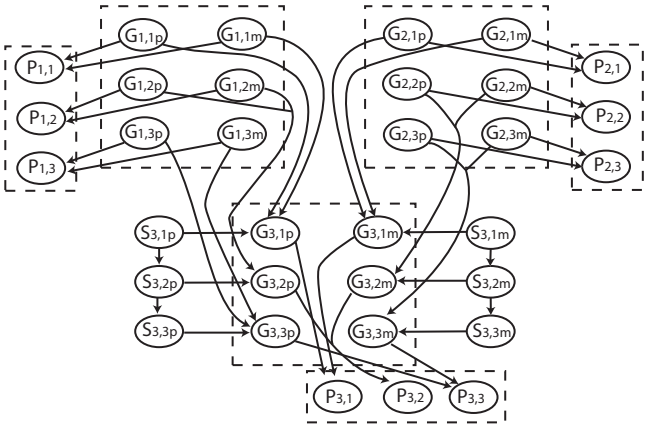


Figure 5.6: Each stippled box holds information for a single individual. Each pair of nodes in a box holds information on a single marker. The genotype of a child is only dependent on the genotypes of the parents, and the selector nodes, which encodes the inheritance patterns.

In the following we look at the inference flow of the Bayesian network in Figure 5.6. To clarify the difference we describe the flow in the two extremes: Elston-Stewart and Lander-Green style inference.

The order with which the nodes are marginalized out of the network, i.e. the elimination order, is found by creating the *moral graph* of the network, triangulating the graph by adding fill-ins and constructing the junction tree for inference calculations. The choice of which nodes should be connected by the fill-ins is described in Appendix B.

A moral graph is created by removing the edge directions in the original network. Then edges are added between nodes with common *graph*-children[3], see Figure 5.7. The edge between $G_{1,1p}$ and $G_{1,1m}$ is one such additional edge.

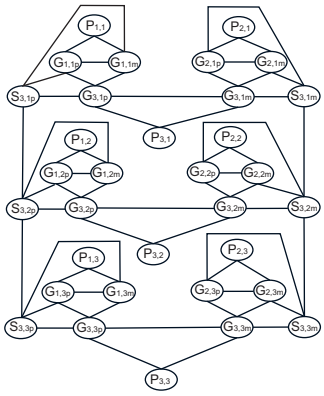


Figure 5.7: The moral graph for the Bayesian network in Figure 5.6.

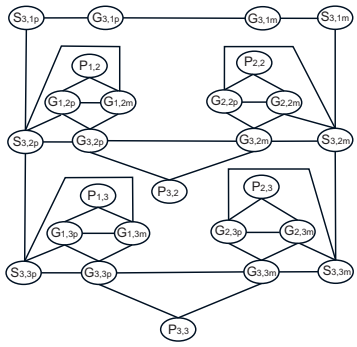


Figure 5.8: The triangulated cliques for marker one is removed. This leaves the two selector variables and the genotype information for individual 3 at the first marker.

5.1 Lander-Green Elimination Order

The Lander-Green branch algorithms eliminates one marker at a time. In Figure 5.8 the triangulated cliques for marker 1 are removed, leaving the two selector nodes $S_{3,1p}$ and $S_{3,1m}$, and the two genotype nodes $G_{3,1p}$ and $G_{3,1m}$.

To eliminate the last four variables at the first marker fill-ins have to be added. The first variables which are eliminated are the genotype nodes, as this adds the fewest fill-ins, and as the interesting[4] nodes are the selector nodes. The genotype nodes can be eliminated in any order. In Figure 5.9 the node $G_{3,1p}$ is the first to be removed. This creates a fill-in between $S_{3,1p}$

[3]We have to differentiate between children and parents in the pedigree, and the expression of talking about child- and parent nodes of a graphical structure.

[4]With respect to linkage analysis.

and $G_{3,1m}$. Then the next genotype node $G_{3,1m}$ is removed, adding a fill-in between $S_{3,1p}$ and $S_{3,1m}$, see Figure 5.10.

This leaves the two selector nodes on the first marker. As for the genotype nodes, it does not matter in which order the selector nodes are eliminated. In Figure 5.11 the paternal selector node $S_{3,1p}$ is removed first. This creates a fill-in between the paternal selector node on the second maker $S_{3,2p}$ and the maternal selector node on the first marker $S_{3,1m}$. Only one node is left for the first marker. Elimination of $S_{3,1m}$ creates a fill-in between the two selector nodes on the second marker $S_{3,2p}$ and $S_{3,2m}$, see Figure 5.12. The nodes at the other markers are eliminated in a similar fashion.

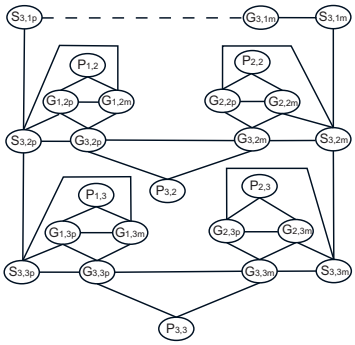


Figure 5.9: The first genotype node $G_{3,1p}$ is removed.

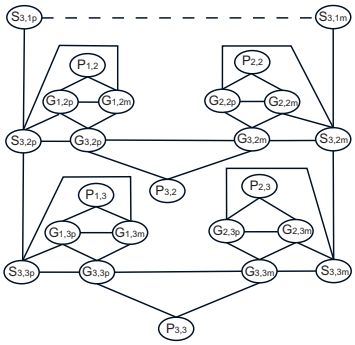


Figure 5.10: The second genotype $G_{3,1m}$ node is removed.

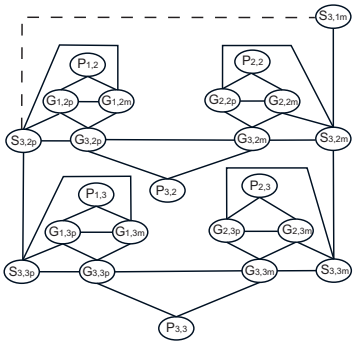


Figure 5.11: The first selector node $S_{3,1p}$ is removed.

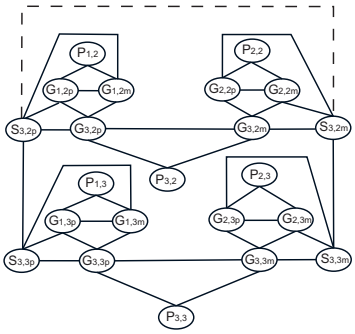


Figure 5.12: The second selector node $S_{3,1m}$ is removed.

The junction tree generated by this triangulation is shown in Figure 5.13. Comparing this with the junction tree for single point, there are some similarities.

The junction tree has the same basic structure for all three markers. In a way it preforms the same operation three times, one for each marker. The three subtrees marked by dotted lines in Figure 5.13, preform the same operation repeated once for each marker. The nodes outside the dotted lines

are nodes for selector node elimination; basically updating one marker with information from another marker.

In Section 4.3 we claimed that the Lander-Green algorithm is exponential in the number of people in the pedigree. Looking at the junction tree it is clear why. For each marker added to the Lander-Green algorithm, the junction tree gets an extra subtree. If another person is added to the pedigree then two nodes are added to each of the subtrees for the markers.

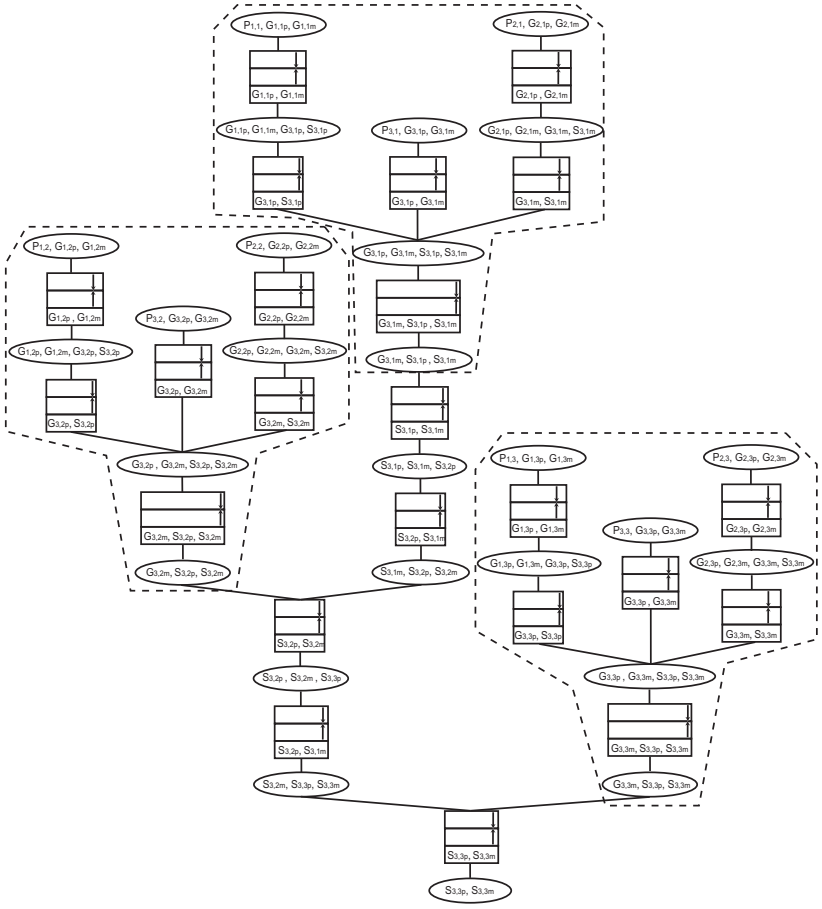


Figure 5.13: A the junction tree for the Bayesian network in Figure 5.6, using the Lander-Green type elimination order.

5.2 Elston-Stewart Elimination Order

Variable elimination in the Elston-Stewart type algorithms are very different from the variable elimination in the Lander-Green type algorithms. In Elston-Stewart the variables are eliminated for one person at a time. Again

we use the moral graph in Figure 5.7 for illustration of the elimination order.

First step is to eliminate person p_1. This is relatively easy as p_1 is a founder and thereby the nodes are triangulated without adding fill-ins, see Figure 5.14. The same goes for person p_2 in the pedigree, see Figure 5.15.

Now there is only one person left in the graph. Person p_3 is a non-founder, and the nodes are not triangulated, except for the phenotype nodes, which can be easily eliminated. See Figure 5.16.

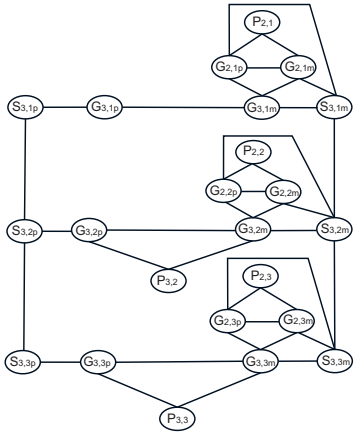


Figure 5.14: The moral graph after eliminating the nodes for person 1.

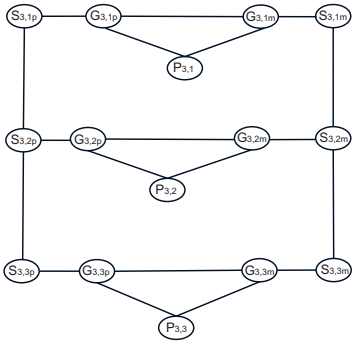


Figure 5.15: The moral graph after eliminating the nodes for person 2.

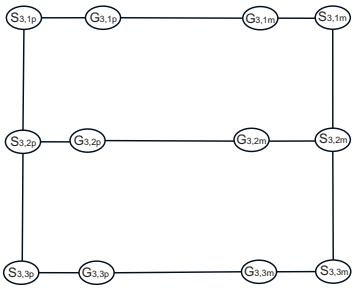


Figure 5.16: The phenotype nodes for person 3 are eliminated.

For the selector nodes and genotype nodes of p_3, fill-ins must be added. We are interested in "pushing" all the information down on the selector nodes for each marker. Therefor it makes sense to eliminate all the genotype nodes first. The fill-ins created are shown in Figure 5.17.

The only nodes remaining are the selector nodes. The fill-ins needed for eliminating these are shown in Figure 5.18, however the fill-ins are here added at random. Dependent on the size of the probability tables at each selector node, other fill-ins might be more optimal.

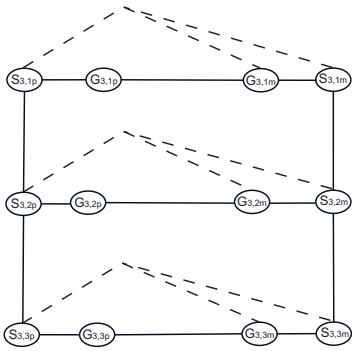


Figure 5.17: The fill-ins needed to eliminate the genotype nodes of person 3.

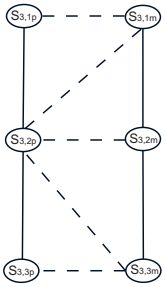


Figure 5.18: The fill-ins needed to eliminate the last nodes in the graph.

The resulting junction tree for the Elston-Stewart type inference flow can be seen in Figure 5.19. The Elston-Stewart junction tree consists of relatively few but very big cliques. For each person there are two nodes in the junction tree. One node for the elimination of the phenotypes and one node eliminating the genotype variables. When adding another marker to the Elston-Stewart algorithm no new nodes are added to the junction tree, the cliques in each node only grow larger. Therefor is the algorithm exponential in the number of markers, as claimed in Section 4.2. For each new person added to the pedigree, two new nodes will be added to the junction tree.

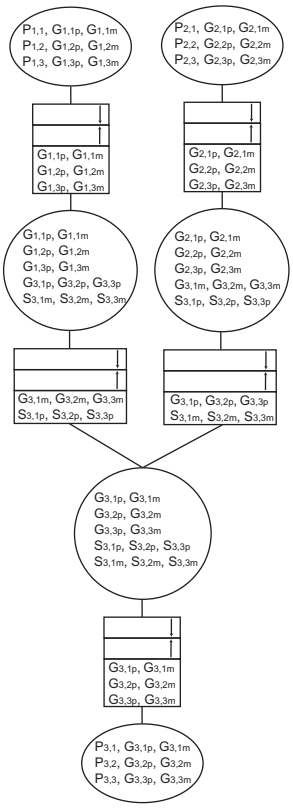


Figure 5.19: A the junction tree for the Bayesian network in Figure 5.6, using the Elston-Stewart type elimination order.

Summary

There are generally two sets of linkage analysis approaches. One is exponential in the size of the pedigree and linear in the number of markers; the other vice versa.

Looking at the two junction trees for the multi point algorithms the differences are striking. The Lander-Green tree is very large but with small joint tables, the Elston-Stewart is small with large joint tables. There are however some similarities. They both eliminate the phenotype and founder nodes as one of the first steps of the algorithm. This makes sense as these are the nodes which are triangulated in the original moral graph without additional fill-ins. Then they remove the genotype nodes and last the selector nodes.

The major difference in the elimination order is seen with respect to the selector nodes. The joint distributions of the inheritance patterns are given by *pushing* the values of the rest of the graph into the selector nodes. In Lander-Green type algorithms two selector nodes are marginalized out together with the rest if the nodes tied to a marker. In Elston-Stewart type algorithms all the other nodes are marginalize out, leaving the selector nodes for last.

6 Probabilistic Decision Graphs

Probabilistic Decision Graphs (PDG) were developed by Marius Bozga and Oded Maler [2], and further developed into Real Function Graphs (RFG) by Manfred Jaeger [18].

The following is an introduction to PDGs together with a description of the operations we need to create our linkage analysis algorithm.

Bayesian networks and BDD-based representation frameworks of probability distribution are developed with similar goals: to obtain compact representations of probability distributions on which certain basic operations can be performed efficiently [18]. BDDs were however created for equality testing, and Bayesian networks and PDGs were created specifically for probabilistic inference problems, which is why we at first got the idea that PDGs could provide a better data structure for linkage analysis than MTBDDs, which are used in the Allegro linkage algorithm in Section 4.4.

The purpose of PDGs is a probabilistic system, which have some of the same properties as Binary Decision Diagrams (BDD), see Appendix C. The desired properties of BDDs are:

1. Canonic - there exist a unique BDD representation for each boolean function.
2. Relatively efficient algorithms for manipulating BDDs
3. Performs well in the analysis of many structured systems: the size of the BDD remains small relative to the size of the state-space.

6.1 Real Function Graphs

Manfred Jaeger introduced a generalization of PDGs called Real Function Graphs (RFG), and we find that his definition of RFGs is a good basis for the definition of PDGs.

Definition 7 *Let $F = \{T_1, \ldots, T_k\}$ be a forest over $\mathbf{X}$, i.e. each T_j is a rooted, directed tree whose nodes are a subset of $\mathbf{X}$, and the union of all nodes in the T_j is $\mathbf{X}$. Let E_F denote the edge relation in F. A* Real Function Graph Structure *for $\mathbf{X}$ with respect to the forest F is a rooted directed acyclic graph $G = (V, E)$, such that*

- *each node $v \in V$ is labelled with a variable $X_i \in \mathbf{X}$.*
- *For each node v labelled with X_i, each $x_{i,h} \in R(X_i)$, and each $X_j \in \mathbf{X}$ with $(X_i, X_j) \in E_F$ there exists one edge e (labelled with $X_{i,h}$) in E leading from v to a node $v' \in V$ labelled with X_j.*

Definition 8 *A Real Function Graph Structure is turned into a* Real Function Graph (RFG) *if*

- *each node v labelled with X_i also is labelled with a* value vector $\mathbf{p}^v = (p_1^v, \ldots, p_{k_i}^v) \in R^{k_i} (i = 1, \ldots, n)$.

We denote the resulting RFG with $G = (V, E, \mathbf{p})$.

Definition 9 *A RFG G is called a* probabilistic decision graph (PDG) *if*

- *for all nodes v with label X_i : $p_h^v \in [0, 1]$ and*
- $\sum_{h=1}^{k_i} p_h^v = 1$.

The variable structure or variable order of a PDG is given by an underlying tree structure T. As an example see Figure 6.1, which is the variable structure of the PDG in Figure 6.2. The structure of a PDG encodes certain (conditional) dependency relations in the distribution P_G. Different PDG structures (for a given F) encode different probabilistic models, i.e. the sets of probability distributions that can be represented over the structures are different.

6.2 Linkage Operations on RFGs

In this section we will give a description of the PDG / RFG operations we need for the linkage analysis algorithm. The operations needed are multi-

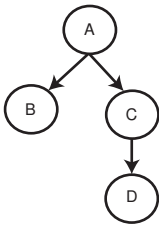


Figure 6.1: The tree structure T over the variables A, B, C, D, with edge relations $E_T = \{(A, B), (A, C), (C, D)\}$.

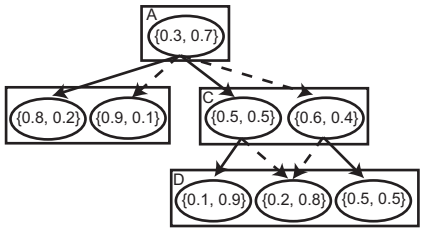


Figure 6.2: A binary PDG over T. A dotted line $= 0$, a full line $= 1$.

plications of various kind, normalization, marginalization and finding the maximum configuration.

6.2.1 Multiplication

When performing any sort of multiplication on RFGs and PDGs, the result is an RFG. In the example figures of this section the RFGs given are actually PDGs, to illustrate that the multiplication operations are applied in the same fashion to the specialized form of RFGs.

Multiplying a RFG with a constant, each entry in the value vector of the root is multiplied with the constant, see Figure 6.3. The probability of each path in an RFG is found by multiplying all the values of the path, and multiplying with a constant is the same as multiplying each path with the constant. It is therefor only necessary to multiply the constant to the root values. This operation is of time complexity $\mathcal{O}(k)$, where k is the number of possible values of the root variables.

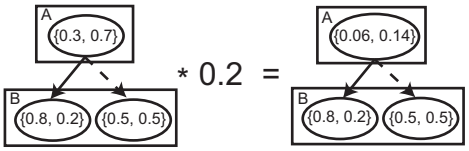


Figure 6.3: When multiplying a constant to a RFG, the constant is multiplied to each value of the root node.

When multiplying two RFGs of the same underlying tree structure T, the multiplication is a recursive function starting at the root. For each variable node in T the vector values at the same positions of the vectors are multiplied, and the child nodes of the corresponding outgoing edges in the two RFGs are multiplied and so forth, see Figure 6.4. This operation is of quadratic time complexity in the size of the (largest) RFG, $\mathcal{O}(n \cdot e) \sim \mathcal{O}(n^2)$ if $n > e$. The size of an RFG is the number of edges in the RFG.

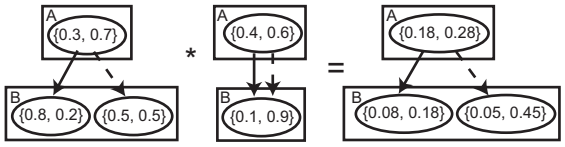


Figure 6.4: When multiplying two RFGs, the values of each corresponding node pair are multiplied.

When multiplying two RFGs with the same general tree structures, differing only in missing variable nodes, first dummy nodes are added to represent the missing nodes and thereby create identical tree structures. The resulting RFGs are then multiplied as above, see Figure 6.5. The operation of adding dummy nodes is of quadratic complexity $\mathcal{O}(v \cdot w)$, where v is the size the RFG and w is the number of variables to be added. This makes the multiplication operation of time complexity $\mathcal{O}((v \cdot w) \cdot (v' \cdot w')) \sim \mathcal{O}(n^4)$, where v', w' is the corresponding variables in the other RFG.

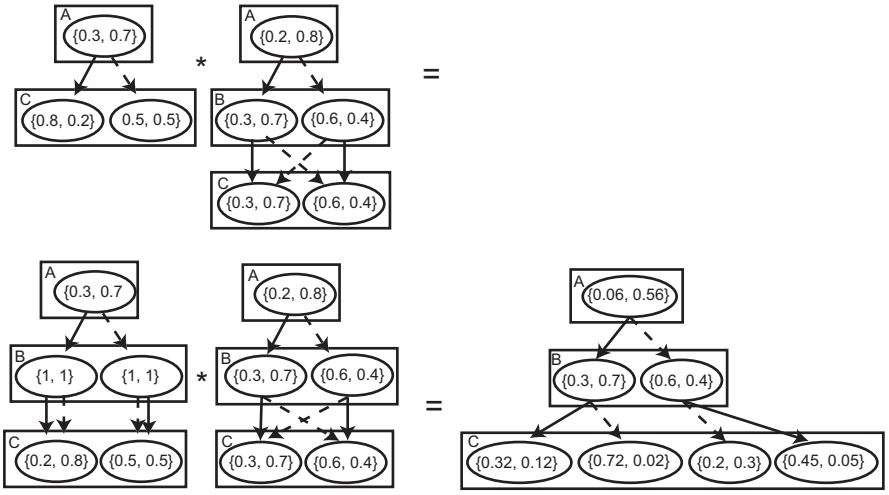


Figure 6.5: When multiplying two RFGs (or PDGs) with different tree structures, first dummy nodes are added to create identical structures, then the trees are multiplied as above.

6.2.2 Max Configuration

The maximum configuration of an RFG / PDG, i.e. the path which has the highest probability, is found by:

$$\max_{A,B,C,D} F^{\vartheta} = \max_{A} val(A) \cdot \max_{B,C,D} F^{\vartheta'}$$

where F^{ϑ} is an instantiation of the graph, i.e. a path in the graph, with the underlying tree structure T of variables A, B, C, D.

The result given is in general the probability of the max configuration. Finding the maximum configuration of an RFG is done through one sweep of the graph, and is therefor of linear time complexity $\mathcal{O}(e)$, where e is the number of edges in the graph.

6.2.3 Marginalization

When marginalizing out a variable node X in the tree structure T, from a RFG the value-nodes above ϑ are left untouched. We can therefor think of each we think of each value-node ϑ of X as being the root node, which we want to remove. For clarification we depict the RFG as a tree in Figure 6.6.

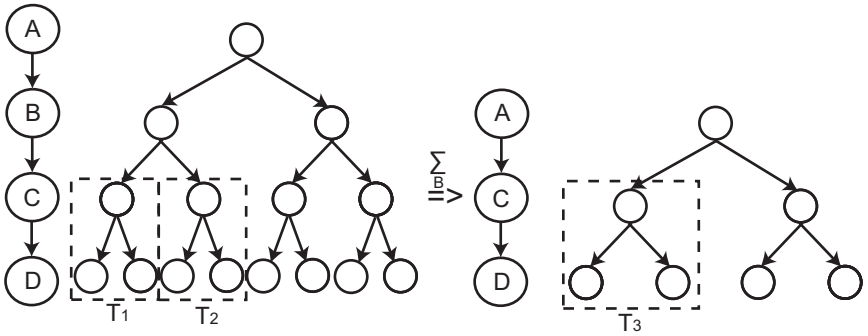


Figure 6.6: When marginalizing out a node, each subtree under the node is weighted by the corresponding value in the node. Then the subtrees are combined by addition.

When marginalizing out a root node, the subtrees are multiplied in the same fashion as when multiplying two PDGs of the same tree structure. Se previous Subsection 6.2.1 on multiplication. However, the general function is as follows:

$$T_3 = v_1 \cdot T_1 + v_2 \cdot T_2$$

Where v_1, v_2 are the values of the root node which combines the two subtrees T_1, T_2.

When multiplying two nodes the resulting RFG is dynamically reduced, such that resulting nodes with identical values are merged. In regular multiplication if a node θ_1 is to be multiplied with another node θ_2, and both θ_1 and θ_2 are on several paths of their respective RFGs, the multiplication is only done once, and at all other times the first resulting node is set. In marginalization however the two nodes will be weighted, quite possibly by different weights for each multiplication, thereby resulting in different resultant nodes. To quote Manfred Jaeger in [18], on marginalization in PDGs:

> While algorithmically not very difficult, this procedure can cause an exponential blowup in the size of the PDG ...

6.2.4 Normalization

An RFG can be turned into a PDG, by normalizing the value vector of each node. Normalization of the nodes are calculated not only on the basis of the values of the individual nodes, but also on the *outflow* (ofl) of in the child nodes.

$$v_{i_{norm}} = \frac{v_i \cdot ofl(ch_{v_i})}{\sum_{j=1}^{k} v_j \cdot ofl(ch_{v_j})}$$

The outflow of a node ϑ is given by adopting the node as a root, and summing over all the possible values of the subtree, see Figure 6.7. In a PDG the outflow of each node is 1. Normalization of a RFG is done in one sweep of the graph, and is thereby of linear time complexity $\mathcal{O}(e)$, where e is the number of edges in the graph.

$$ofl(\vartheta) = \sum_{X,Y,Z,U} all\ F^{\vartheta}$$

where $F^{\vartheta} = (X = x, Y = y, Z = z, U = u)$ is an instantiation of the subtree consisting of the four variables X, Y, Z, U.

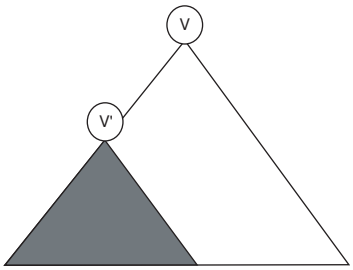


Figure 6.7: The outflow of a node ϑ is given by summing over the values of the subtrees for which ϑ is root.

Summary

This chapter was introduction to Probabilistic Decision Graphs (PDG) together with a description of the operations, which we must apply to these to create a linkage analysis algorithm using PDGs.

The purpose of PDGs is a probabilistic system, which have the same properties as Binary Decision Diagrams (BDD).

Real Function Graphs (RFG) are a generalized form of PDGs. In PDGs the value vector of each node sum up to one, i.e. $\sum_{i=0}^{|V|} v_i \in V_n = 1$, where V_n is

the value vector of node n. In RFGs there are no constraints on the values which can be saved in the nodes.

The operations we need for the linkage analysis algorithm are:

- multiplication,
- normalization,
- marginalization and
- maximum configuration.

7 PDG Linkage Algorithm

In this chapter we will describe our linkage algorithms and the data structure used. The algorithm is a Lander-Green type algorithm, see Section 4.3, which means that first single point analysis is done at each marker, and multi point analysis is done by updating the probabilities of the sets of inheritance patterns at each marker, with respect to the neighboring markers.

The goal was to create an optimized version of DeCode's FASTTREETRAVERSAL Algorithm using probabilistic graphical models; we have chosen PDGs. Therefor is our algorithm conceptually very close to this algorithm, which is described in Section 4.4. Besides using a different data structure for storing and manipulating the sets of inheritance vectors and their probability distributions, we have made a few additional changes.

We have incorporated a few preprocessing steps inspired by Superlink. These steps helps create more and ordered genotype information deduced from the original input data. The new information would originally be found as a part of the linkage analysis, but with preprocessing some invalid inheritance patterns are discovered earlier, which reduces the size of the PDG.

The changes we have made to the FASTTREETRAVERSAL Algorithm are due to the use of PDGs instead of MTBDDs, except for one case. We have expanded the concept of components in the set $\mathcal{E}$ of ambiguously assigned founder alleles.

7.1 Preprocessing

Before doing the actual linkage analysis, we preprocess the pedigree to get as much genotype information as possible. As mentioned in Chapter 3 the genotype information is often filled with holes. This is one of the major reasons for exponential blowups of the algorithms, as no inheritance patterns are impossible when no genotype information is given. However often times many of the holes can be filled by deducing the possible genotypes from the given information. More genotype information given the algorithm means less possible inheritance patterns, and earlier detection of incompatible inheritance patterns. It is possible to order or even fill in genotype information for the following cases:

- If both parents and the child have full genotype information, and they do not have the same heterozygous genotype. See Figure 7.1.
- Given a child with parents (p_1, p_2), with full genotype information on p_1 and either none or partial information on p_2. If only one value of the child's genotype matches a value of p_1, the genotype of the child can be ordered, and additional additional information might be assigned to p_2. See Figure 7.2
- If the child is homozygously assigned, the genotype is already ordered, and each of the parents can be assigned additional genotype information, if this is missing. See Figure 7.3.
- If one or both parents are homozygously assigned, and the child is given no genotype information. Genotype[1] information on the child can be deduced. If both parents are homozygous the information on the child will even be ordered. See Figure 7.4.
- If there is partial genotype information on the child, and one of the parents p_1 is homozygously assigned. If the information on the child does not match the information on p_1, the child can be given full genotype information. If the other parent p_2 is given genotype information, the genotype of the child can be ordered. If p_2 is missing genotype information, it might be updated based on the new information given the child. See Figure 7.5
- If there is no information on the child, the child is a leaf node and the parents are heterogenous the child can be removed. This is done in a recursive manner, such that all uninformative individuals are trimmed from the pedigree. This operation is similar to one of the pre-processing steps in the Superlink algorithm. See Figure 7.6.

[1] Or phenotype in the case of only one parent being homozygous.

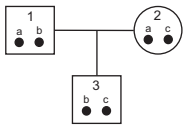


Figure 7.1: The genotype information on the child, person number 3, can be ordered, because the only possible inheritance pattern given the genotype information is such that person 3 inherits the b allele from his father and the c allele from his mother.

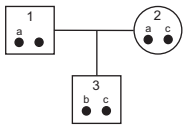


Figure 7.2: The genotype information of person 1 can be updated, because the information given on persons 2 and 3, is such that the missing allele must be a b.

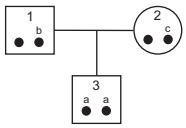


Figure 7.3: The information given on the child, person 3, can update both the information given on both the mother and the farther.

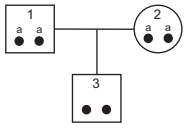


Figure 7.4: Both parents are homozygous (they could have been homozygous, but of different genotype). The genotype of the child can be deduced and ordered.

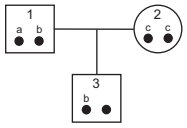


Figure 7.5: The information given on parents and child, makes it clear that the missing allele on the child is a c passed from the mother, and that the given b was passed from the father.

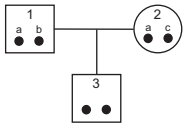


Figure 7.6: A leaf child (with no children of it's own) can be removed when it has not been given genotype information. It will not add any new constraints on the possible inheritance pattern, and can therefor be ignored.

7.2 Linkage PDG Structure

As in the FASTTREETRAVERSAL Algorithm the underlying tree structure, or variable order, is actually a sequence of the non-founders of the pedigree. The non-founders are added to the sequence such that both parents are either founders or already added to the sequence. The order of the non-founders is the same for each marker.

We represent sets of inheritance vectors at each marker as a PDG, where each node has four outgoing edges $\{mm, mp, pm, pp\}$ with a corresponding vector of probability values. Each of the nodes are labelled with a set of compatible founder allele assignments $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ as described in Section 4.3. We denote a $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ set compatible with an inheritance vector v, $\mathcal{F}(v)$ for short.

For each child node v' of the parent node v, the set $F(v')$ is a subset of $F(v)$, because more constraints have been added given the genotype information of the child, and thereby there are less compatible founder allele assignments. I.e. the set of compatible founder allele assignments $\mathcal{F}$ will

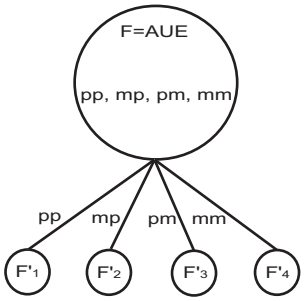


Figure 7.7: An example linkage PDG for one non-founder. The leaf nodes are dummy nodes, holding only the $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ set of the inheritance vector including the final non-founder.

be smaller the further towards the leaf nodes of the PDG. Therefor can we build the probability distribution of the PDG in a top down fashion by

$$P(\mathcal{F}(v')) = P(\mathcal{F}(v))P(\mathcal{F}(v')|\mathcal{F}(v))$$

The major difference between using MTBDDs and PDGs, is that in the MTBDDs of the FASTTREETRAVERSAL Algorithm the structure is boolean, with the probability of each finished path given in the terminal node. Whereas in PDGs the probability of a path is given by multiplying the probability values on the path.

7.2.1 Probability Calculation

Generally for all the algorithms the probability of an inheritance pattern is calculated as the probability of the consistent founder allele assignments given the inheritance pattern. Several inheritance patterns will however share common consistent founder allele assignments. The probability of a specific inheritance pattern being the resulting factor in a founder allele assignment, will therefor only be a fraction of the probability of the assignment, see Figure 7.8.

Example: Say we have four founder alleles $\{f_1, f_2, f_3, f_4\}$, and two inheritance patterns v and v', where $\mathcal{F}(v) = \{f_1 \xleftrightarrow[a]{A} f_2, f_3 = \emptyset, f_4 = A\}$ and $\mathcal{F}(v') = \{f_1 = A, f_2 = a, f_3 \xleftrightarrow[a]{A} f_4\}$. The consistent founder allele assignments are shown in Table 7.1, where the identical assignments are highlighted.

The probability we really are interested in is the probability of an inheritance pattern given the genotype information, $P(v|\mathcal{G})$. This is proportional to a constant to the probability of the set of consistent founder allele assign-

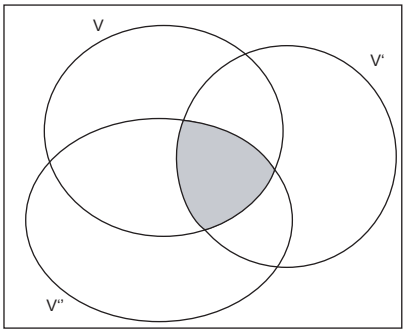


Figure 7.8: A compatible founder allele assignment, can be the result of several inheritance patterns.

$\mathcal{F}(v)$				$\mathcal{F}(v')$			
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4
A	**a**	**a**	**A**	**A**	**a**	**a**	**A**
A	a	A	A	A	a	A	a
a	A	a	A				
a	A	A	A				

Table 7.1: An example of two sets of compatible founder allele assignments that have an assignment in common.

ments $P(\mathcal{F}(v))$ for the given inheritance pattern.

$$\begin{aligned} P(v|\mathcal{G}) &= P(\mathcal{G}|v)P(v) \\ &= P(\mathcal{F}(v))P(v) \\ &= P(\mathcal{F}(v))\mathbf{c} \end{aligned}$$

Where the probability of the genotype given the inheritance pattern is the same as saying the set of compatible founder allele assignments given the inheritance pattern. The probability of the inheritance pattern can be treated as a constant because all inheritance patterns in themselves are equiprobable.

This means that when using the founder allele assignment probability as inheritance vector probability, the probabilities do not sum up to 1, but to a constant and must therefore be normalized when a PDG is needed instead of an RFG.

The probability of the founder allele assignments are calculated on basis of the population frequency, under a couple of assumptions:

1. We assume that all possible genotype combinations are viable (meaning they do not lead to embryos which cannot be born or grow up to adulthood).

2. We assume that the genetic material have been mixed over the population over many generations, such that even though the markers are located relatively close to one another, the different combinations are equally spread over the population.

With this general picture of the calculation of the probability distributions for the algorithm, we proceed to describe the single point construction of the inheritance patterns.

7.3 Single Point Algorithm

There are three general steps in the single point algorithm for finding the most probable inheritance vector at a marker:

1. Create the Real Function Graph Structure (RFGS).
2. Calculate the probability values of the PDG.
3. Find the most probable inheritance pattern.

7.3.1 The Real Function Graph Structure

The basic algorithm used for building the Real Function Graph Structure (RFGS) is the FASTTREETRAVERSAL. The general tree structure T is a sequence of variables. For each non-founder a new variable is added as a leaf to the tree structure. The variable node is populated with rfgs-nodes, where for each rfgs-node in the parent variable, i.e. the old leaf, the founder set $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ is updated with respect to the given genotype information on the non-founder, and the inheritance pattern leading to the node. The RFGS is built and reduced dynamically top-down. Two rfgs-nodes of the leaf variable are merged if they have equal $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ sets and assignment of founder alleles to the *pedigree*-parents of the leaf individual. If nodes full-fill this we know that the subtrees will be identical to. In reality this only guaranties a merge of the rfgs-nodes for invalid paths, i.e. paths leading to incompatible founder allele assignments.

When all the non-founders are included, the RFGS is reduced bottom-up by merging nodes with identical $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ sets and *graph*-child nodes, ignoring the pedigree parent assignments.

7.3.2 The Merge Operation

As mentioned previously we have decided to change the structure of the founder allele components in $\mathcal{E}$ described in FASTTREETRAVERSAL. The

split operation described in Subsection 4.4.1 enforces a maximum component length of 2, but results in multiple copies of the same inheritance pattern. We replace the split operation with a *merge* operation. The merge operation combines the two existing founder components to a single component, by adding an edge between the two implicated founder alleles. This creates chains or graphs of connected founder variables with no size limit, but keeps only one copy of the inheritance pattern.

In the split operation checking whether an allele assignment is consistent is just a look up at the possible values of current founder variable. Whereas in the merge operation it takes linear time to assign the chain of founder variables in set $\mathcal{E}$ and check for compatibility for each assignment. However the split operation creates several copies of the same inheritance pattern, which then must be combined at a later time, where the merge operation saves all the compatible founder allele assignments in one set on the inheritance pattern.

7.3.3 Calculating the Probability Values and the Maximum Configuration

The probability values calculated on basis of the sets of compatible founder allele assignments does not sum up to one, as described in Section 7.2.1. The first result for each marker is therefor a RFG.

The values of the nodes of the RFG are calculated based on the compatible founder sets at the child nodes, in a top-down fashion. When the values have been calculated for the final non-founder based on the founder sets at the dummy leaf nodes, the dummy nodes are discarded.

The RFG is reduced once again, based this time only on the values of the nodes. I.e. in a bottom-up fashion the nodes with identical value vectors and children are merged. The RFG is then normalized to a PDG, as described in Subsection 6.2.4, and reduced again if possible.

The final step in single point analysis is finding the maximum configuration of the PDG, and thereby finding the most probable inheritance pattern at the given marker. The operation of finding the maximum configuration is described in Subsection 6.2.2.

Ordinarily when finding the maximum configuration of a PDG, the result is a probability value. In linkage analysis however it is not enough to know the value, we must also remember the exact configuration of the graph[2], and this additional information must be stored at each node.

If we are doing single point analysis the algorithm is finished, and the output is the maximum configuration of the PDG. If however we are doing

[2]To know the inheritance pattern.

multi point analysis, the maximum configuration operation is not called until the PDGs at each marker has been updated with respect to the PDGs at the neighboring markers.

7.4 Multi Point Algorithm

In the multi point algorithm the PDGs created in the single point analysis are updated with respect to the PDGs at the neighboring markers and the recombination fraction between these. The influence of the recombination fractions are encoded in an RFG which we term the *recombination graph* R_i, where i is the index of the leftmost marker.

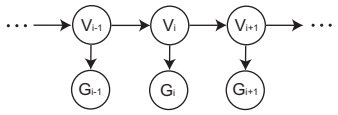


Figure 7.9: The PDGs built at each marker is first updated with respect to each other in a left to right fashion.

The PDGs are updated first left to right, as seen in Figure 7.9, then right to left. The left update between markers M_1 and M_2 is done by first multiplying the PDG for M_1 with the recombination graph R_1, then multiplying the result with the PDG for M_2.

7.4.1 Recombination Graph

The recombination graph is constructed such that the underlying tree structure T_R is a sequence $\{v_1, w_1, v_2, w_2, \ldots, v_n, w_n\}$, where v_i is the non-founder i at M_1 and w_i is the non-founder i at M_2.

For each v_i there is only one pdg-node, and for each w_i there is four, one for each value attribute $\{mm, mp, pm, pp\}$ of v_i. The value vectors of the pdg-nodes are constructed as follows:

- For v_i the value vector is $\{1, 1, 1, 1\}$, because no recombination has occurred between a marker and itself.
- For w_i the value vectors are constructed such that:
  - if no recombination has occurred between the two markers the probability is $(1-\theta)^2$,
  - if one recombination has occurred the probability is $(1-\theta)\theta$, and
  - if two recombinations have occurred the probability is θ^2.

This means that for each of the pdg-nodes for w_i, following the given edge from v_i, the value vectors are:

$$\begin{aligned} mm &= \{(1-\theta)^2, (1-\theta)\theta, (1-\theta)\theta, \theta^2\} \\ mp &= \{(1-\theta)\theta, (1-\theta)^2, \theta^2, (1-\theta)\theta\} \\ pm &= \{(1-\theta)\theta, \theta^2, (1-\theta)^2, (1-\theta)\theta\} \\ pp &= \{\theta^2, (1-\theta)\theta, (1-\theta)\theta, (1-\theta)^2\} \end{aligned}$$

A recombination graph for one set of v_i, w_i is shown in Figure 7.10. The recombination graphs for each non-founder are combined into one big graph by letting all outgoing edges from the w_i nodes, go to the node v_{i+1}, i.e. the v node for the next non-founder.

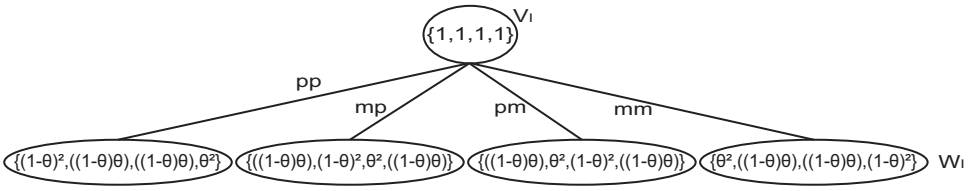


Figure 7.10: A recombination graph for one non-founder.

7.4.2 Updating the PDGs

As described previously, the left recursive update is done by first multiplying the PDG built for marker M_i with the recombination graph R_i, and then multiplying the resulting RFG S_i with the PDG built for marker M_{i+1}. In Figure 7.11 a recombination graph and two neighboring markers are shown.

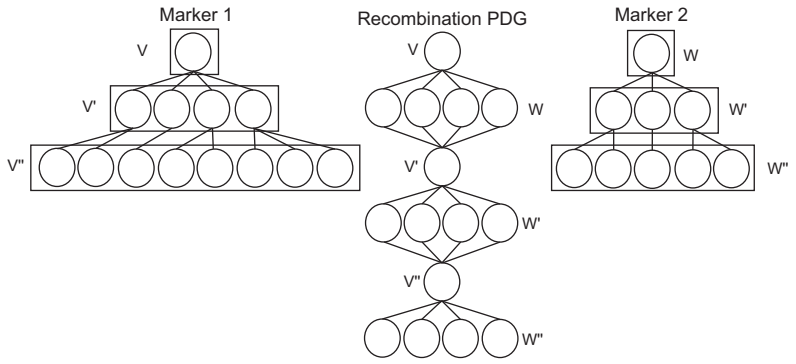


Figure 7.11: Two PDGs for two markers and the recombination graph between them.

The multiplication of the PDG at M_i and the RFG R_i is done as in Subsection 6.2.1. The two tree structures T_{M_i} and T_{R_i}, are such that the variable

order of the variables which they have in common is the same. Therefor the first step is to add dummy nodes to the PDG, see Figure 7.12.

After the multiplication, the variable nodes **v** of the PDG at M_i, are marginalized out of the RFG S_i, as described in Subsection 6.2.3. The resulting S_i' only contains variable nodes of the second marker M_{i+1}. Finally S_i' and the PDG for M_{i+1} are multiplied, resulting in an RFG which is normalized into a PDG, leaving M_{i+1} left-updated. The right-update is performed in a similar fashion. Finally the maximum configuration is found at each marker, as described in Subsection 7.3.3.

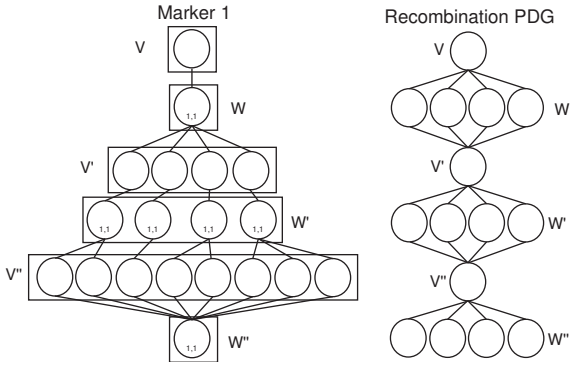


Figure 7.12: Dummy nodes added to the PDG of the first marker, so it can be multiplied with the recombination RFG.

Summary

The algorithm created is a modification of the FASTTREETRAVERSAL Algorithm described in Section 4.4. Instead of MTBDDs, the final output is a set of PDGs, one for each known marker. The underlying tree structure of the PDGs is a sequence of non-founders, ordered such that when one is added the parents are either founders or already in the sequence.

Additional to the algorithm we have incorporated a few preprocessing steps inspired by Superlink, where the genotype information is updated and ordered when possible.

In the basic algorithm the only thing changed is the structure and creation of components in the set of ambiguously assigned founder alleles $\mathcal{E}$, exchanging FASTTREETRAVERSAL's split operation with a merge operation.

Generally the single point algorithm runs in three steps:

1. Create the structure of the PDG. The structure of the PDG is build in the same fashion as described in FASTTREETRAVERSAL.

2. Calculate the probability values of the PDG.

3. Find the most probable inheritance pattern by calculating the maximum configuration of the PDG.

The multi point update with respect to the neighboring markers and the recombination frequency between these, is done by multiplying the PDGs at two neighboring markers and a recombination graph. The PDGs are updated first left to right, then right to left.

8 Implementation

In this chapter we go more in detail with the implementation of the single point algorithm described in the previous chapter. The different parts of implementation includes the format of the input data, the Linkage Java package created and a couple of implementation details from the single point algorithm.

8.1 Filereader

The data read by the Filereader is saved in two files: a pedigree file holding all the information of the individuals of the pedigrees, and a data file holding the allele population frequencies for each marker. These input files are used by other linkage analysis programs such as Superlink. A detailed description of the files can be found on the Superlink homepage [15]. An example pedigree file can be found in Appendix D, this is a data file found at the Superlink homepage. Table 8.1 is a short description of the different data columns in the file, where each row contains the data of one individual.

The file reader reads the pedigree file one line at a time. For every new line it reads it generates a new person object. These objects creates a pedigree graph, which contains all the information given by the input file. The pedigree graph is run through preprocessing as described in Section 7.1.

8.2 Single Point Algorithm

The single point algorithm takes as input the pedigree graph described previously, and as described in Section 7.3. First the RFGS, i.e. the structure, is

Column	Description
1	Holds the id of the pedigree to which the individual belongs.
2	Holds the id of the individual described by this row.
3	Holds the id of the father of the person.
4	Holds the id of the mother of the person.
5	Holds the id of the first child of this person.
6	Holds the id of the next sibling with the same father as this individual.
7	Holds the id of the next sibling with the same mother as this individual.
8	Holds the sex of the person. This is information is only used when looking at sex-linked traits.
9	Does not contain any information in the example file in Appendix D. This column exist because previous linkage algorithms used it for storing system specific data.
$10 \rightarrow$	The rows are paired. Such that row 10 and 11 contain the unordered genotype information on marker 1, 12 and 13 contain information on marker 2 and so forth.

Table 8.1: The different columns in the pedigree data file.

created. Then the values of the RFG are calculated. The RFG is at present time the output of the algorithm.

8.2.1 The Linkage Java package

We write the algorithm in Java, and this subsection is meant as a very short introduction to the main classes used for building the RFGS described in Subsection 7.3.1. The classes for the PDG Linkage Algorithm is a Java package called Linkage. The Linkage Java package includes classes for building the RFGS and a `PDGBuilder` that takes in the RFGS and spits out a PDG. The `PDGBuilder` uses a PDG Java package developed by Manfred Jaeger and Jens Dalgaard Nielsen, at Aalborg University.

The classes included in Linkage package for constructing the RFGS, are `StructTree`, `StructTreeNode` and `StructNode`. The `StructTree` is a RFGS for one marker, consisting of `StructTreeNode`s. The `StructTreeNode`s correspond to the variable nodes of the tree structure of a PDG. Each `StructTreeNode` contains a set of `StructNode`s. These are the actual rfgs-nodes. See Figure 8.1.

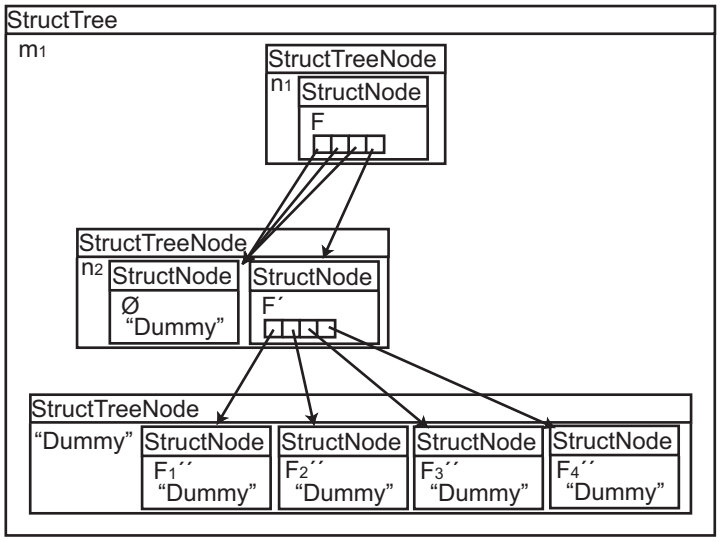


Figure 8.1: The RFGS for marker M_i is build as a `StructTree`, consisting of $N + 1$ `StructTreeNodes`, where N is the number of non-founders in the pedigree. The `StructTreeNodes` are populated by `StructNodes`, where inheritance patterns leading to invalid founder allele assignments are dummy nodes with no descendants.

8.2.2 Founders and Non-founders

The very first step in creating the RFGS is dividing the individuals of the pedigree into founders and non-founders, and creating the first $(\mathcal{A},\mathcal{U},\mathcal{E})$ set.

For each founder, two loci variables (f_m, f_p) are created. If any genotype information was given to a founder, this is assigned to the variables. The given genotype information is unordered, and since there is no way of determining wether one or the other is the maternal or the paternal allele, we assign the genotype information at random. Phenotype information is treated as partial genotype information[1], and only one of the loci variables is then assigned a value. As the founder loci are assigned value they are divided in DIVIDEFOUNDERS into the first set of compatible founder allele assignments $(\mathcal{A},\mathcal{U},\mathcal{E})$, which is the root information for the single point algorithm later in this chapter.

Algorithm DIVIDEFOUNDERS(P, G_F)
1. **for** each $f \in F$ with alleles $\{a_{1f}, a_{2f}\}$ and unordered genotype $\{g_{1f}, g_{2f}$
2. **do if** $g_{if} == 0$
3. $\mathcal{U} \leftarrow a_{if} = 0$
4.

[1]We can do this because we only are working with Mandelian traits, i.e. single locus traits.

5. **if** $g_{if} == x$
6. $\mathcal{A} \leftarrow a_{if} = x$
7.

The $(\mathcal{A}, \mathcal{U}, \mathcal{E})$ sets are held in `FounderSet` objects. Each `StructNode` is labelled with such an object, which hold the constrains of the compatible founder allele assignments given all the founders and the non-founders which come before this in the variable order. The sets of compatible founder allele assignments are updated as described in Section 4.3, substituting the split operation with the merge operation described in Subsection 7.3.2. When all the non-founders are included in the RFGS, a dummy variable is added containing only the compatible founder allele assignment sets, for the last non-founder.

8.2.3 PDGBuilder

As mentioned previously the `PDGBuilder` takes the RGFS as input, creates a RFG by calculating the probability values based on the compatible founder allele sets in the rfgs-nodes, as described in Subsection 7.2.1. The RFGS is not mapped directly to the RFG. All the dummy nodes are removed, they only exist to calculated the corresponding values of their parents. The dummy nodes corresponding to invalid sets result in a probability value of zero, and the corresponding out edge is lead to a random child node. See Figure 8.2.

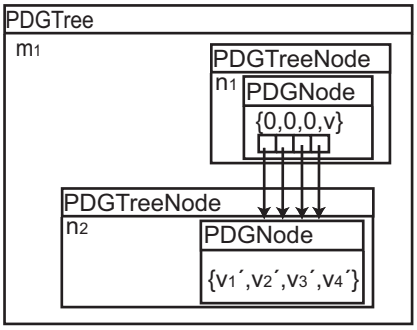


Figure 8.2: The resulting PDG given by the `PDGBuilder` when this has been given the RFGS in Figure 8.1. The PDG for marker M_i is build as a `PDGTree`, consisting of N `PDGTreeNode`s, where N is the number of non-founders in the pedigree. The `PDGTreeNodes` are populated by `PDGNodes`, where the probability values of inheritance patterns leading to invalid founder allele assignments are zero.

8.2.4 Example Test Run

The following is a description of the pedigree input data we have given our linkage analysis program. The pedigree data file can be seen in Appendix D. The data files can be found at the Superlink homepage [15].

The data file only holds one pedigree consisting of 57 individuals, where 15 are founders and 42 are non-founders. The pedigree is graphically represented in Appendix D Figure D.1.

The RFGS built by the algorithm given this input data consists of 184 nodes, mapping to a RFG of 105 nodes. In the first version of the algorithm we only merged the invalid nodes, such that only one such existed for each variable node, but all valid founder sets resulted in a new rfgs-node. This created a structure too big to handle. By the 30th non-founder the number of nodes was in the 3-thousands and the computer was out of memory. If instead of FASTTREETRAVERSAL we had implemented LANDER-GREEN, the size of the RFGS would be 4^50.

Unfortunately we have not been able to get more data for testing, or run some of the other linkage algorithms to do space and time comparisons. The only result our test has provided is that our single point algorithm works, and that it is of reasonable size compared to the input.

8.3 Optimization

The current implementation of the single point algorithm is very basic. Many of the optimizations described in Section 7.3, have been omitted due to time constraints. First of all the only reductions of the different graph structures is the dynamic reduction at creation time, and the reduction of the RFG. This implementation have given us experience in programming these types of algorithms, but most important it gave us a feel of how big the RFGS becomes. The size of the RFGS is the bottleneck of the entire algorithm, because this is the largest graph the algorithm builds. The intermediate reductions reduces the number of computations to perform, however the size of the final PDG will be the same with or without those reductions.

Another omission of the implementation is normalization of the RFG into a PDG. Therefor is the return value of the `PDGBuilder` a RFG, and not the PDG that a multi point algorithm would need as input.

Besides the omissions of optimizations included in the algorithm as it was described in Chapter 7, there exist a couple of other ways of reducing the intermediate graphs even further. These optimizations include checking for founder set equality, reduction in the number of pedigree parent assignments to remember and changing the variable order from a sequence

to a tree.

8.3.1 Founder Set Equality

As described in Section 7.3 the RFGS is twice reduced with respect to the sets of compatible founder allele assignments. First dynamically at creation time, where the nodes are merged both with respect to equality of founder sets and pedigree parent assignments, second when the entire RFGS is build the pedigree parent assignments are ignored, and the graph is merged bottom-up based only on equality of the founder allele assignments.

The equality of sets of founder allele assignments is one problem to which we have not found a satisfying solution. At the moment equivalence is tested by creating a `String` of all the founder alleles and their assigned values. The information in the `String` is ordered, and in this fashion if two sets are exactly identical the `String`s are identical. However this solution is not optimal, because we only find the identical sets and not the equivalent sets.

The idea behind founder set equality is intuitively the same as Founder Reduction described by Daniel Gudbjartsson in [12]. The idea is that consistently changing the allele pointing to a founder, from paternal to maternal and vice versa for all children, this will give an inheritance pattern of the exact same probability of the original. This is due to it being impossible to distinguish between the maternal and the paternal allele of the founders. Daniel Gudbjartsson created the FASTTREETRAVERSAL and when describing Founder Reduction he was thinking in terms of binary inheritance vectors and MTBDDs. With PDGs creating a normal form for testing the equality of the sets of consistent founder allele assignment, more nodes will be merged, reducing the number of calculations.

8.3.2 Reduction by Pedigree Structure

As mentioned above when the RFGS is first created it is dynamically reduced based partly on the founder allele assignments to the pedigree parents.

We can minimize the number of pedigree parent assignments by incorporating the original pedigree structure. The sequential variable order of the non-founder means that there are dependencies between non-founders which are in different subgraphs of the pedigree. Whenever a rfgs-node is created for a non-founder an array of four pedigree parent assignments are created, corresponding to the edges $\{mm, mp, pm, pp\}$. These are the pedigree parent assignments are saved at each rfgs-node of the graph children of the current node. If the current node is a leaf node in the pedigree, the

graph-children can be the roots of sub-graphs with identical sets of compatible founder allele assignments, and the only difference being the pedigree parent assignment of the current node. This is an inconsequential difference, as the assignment will not be used again, the node being a leaf with no decedents.

The founder allele assignments to pedigree parents are reduced based on the structure of the pedigree, such that leafs of the pedigree are never included, and parents where all children have been analyzed are removed. This means that the only assignments saved, are assignments on pedigree-parents with descendants still to be analyzed.

8.3.3 Variable Ordering

Another way of exploiting the pedigree structure to decrease the number of pedigree parent assignments to be remembered, is to change the variable order, or the tree structure T of the PDG, where instead of using the sequence of non-founders used by FASTTREETRAVERSAL, we create an tree structure of the non-founders more like the pedigree structure, see Figure 8.3. This would also take advantage of the PDGs, because they were created to model dependencies between variables, and with the sequence we are creating dependencies which did not exists in the original pedigree.

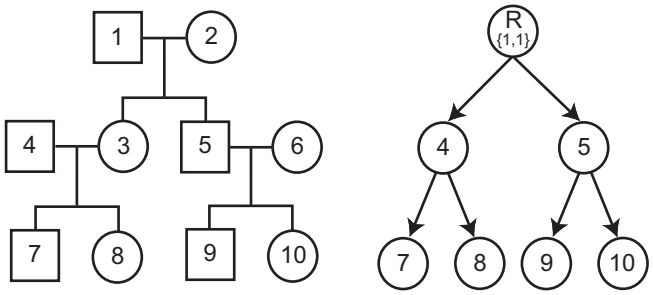


Figure 8.3: An alternative tree structure for the PDG linkage algorithm. This structure would map closer to the pedigree and thereby we would not need to manually control which pedigree parent assignments to remember.

However pedigrees containing loops will be a problem. Loops are created for instance by cousins marrying each other. The tree structure of the RFG must be a acyclic graph, and therefor a strategy for handling such loops must be figured out.

Summary

In this chapter we have described some of the details of the implemented single point algorithm together with a description of the format of the input data, the Linkage Java package created and a couple of optimization ideas for further development.

The data is given in two files: a pedigree file holding all the information of the individuals of the pedigrees, and a data file holding the allele population frequencies for each marker.

The single point algorithm takes as input a pedigree graph created from the input data. First the RFGS, i.e. the structure, is created and the values of the RFG are calculated. The RFG being the output of the single point algorithm at present time.

The RFGS built for our test data consists of 184 nodes, resulting in a RFG of 105 nodes.

The current implementation of the single point algorithm is very basic. Most importantly the only reductions of the different graph structures is the dynamic reduction at creation time.

9 Conclusion

For two semesters we have worked with linkage analysis. In the first semester we gathered information to understand the problem of linkage analysis. In the second semester we have worked on creating our own linkage analysis program using PDGs.

Linkage analysis can be divided into three steps: The probabilistic step, where the probability distribution for the inheritance patterns of the markers is found. The LOD score calculation, where the linkage with respect to the trait under investigation is calculated, and the evaluation step, where the most probable areas of the DNA string are picked for further analysis. Generally algorithms for doing linkage analysis only perform the probabilistic step, and as output returns the most probable inheritance pattern(s) for each marker.

Many different algorithms for linkage analysis exists. We chose four for further investigation: the FASTTREETRAVERSAL as this is the algorithm we wished to optimized, Superlink because it uses Bayesian networks, which is a graphical model, and the Elston-Stewart and the Lander-Green algorithms because these are fundamental algorithms, upon which the other two are based. The algorithms have been examined closely both with respect to the details of the algorithms and their inference flows. Through the understanding gained by this examination we have defined linkage analysis to be a function $\mathcal{L}(P, \mathcal{G}(\mathbf{M}))$, taking a pedigree P and genotype information for the individuals in the pedigree as input, and giving a set of most probable inheritance vectors for the set of markers $\mathbf{M}$ as output.

We have created both a single point and a multi point linkage analysis algorithms using PDGs. Conceptually the algorithms are very close to the FASTTREETRAVERSAL Algorithm developed by DeCode, however changing the data structure of course changes the actual implementation. The FASTTREETRAVERSAL is a Lander-Green type algorithm, which means it

first does single point analysis, and then multi point analysis is done by updating the probability distributions of the inheritance patterns with respect to neighboring markers.

We implemented the single point algorithm and Superlink inspired preprocessing. The preprocessing orders and fills in holes in the genotype information, when possible. This creates more genotype information and reduces the number of nodes in the PDG, because the inheritance patterns leading to invalid founder allele assignments are discovered earlier than they would have without preprocessing.

Running the single point algorithm on example data from the Superlink homepage, containing a pedigree of 42 non-founders, the size of the output RFG is 105 nodes. This final graph could possibly be reduced even further by merging nodes with the same probability vector values and child nodes. The largest structure build is the Real Function Graph Structure, which is of 184 nodes for the given data. Unfortunately we have not been able to compare the new algorithm with the original FASTTREETRAVERSAL, but it must be said to be a fair optimization of the LANDER-GREEN Algorithm, as this would have built a structure of $4^4 2$ nodes containing all the possible inheritance patterns.

A Publicly Available Linkage Analysis Tools

ASPEX	A set of programs for performing multi point exclusion mapping of affected sibling pair data for discrete traits. [14]
	ftp://lahmed.stanford.edu/pub/aspex/
CRI-MAP	For rapid, largely automated construction of genetic linkage maps, generates LOD tables, and detects data errors. [11]
	http://compgen.rutgers.edu/multimap/crimap/
EH	A linkage utility program to test and estimate linkage disequilibrium between different markers or between a disease locus and markers. [35]
	ftp://linkage.rockefeller.edu/software/eh/
FASTLINK	A faster version of the general pedigree program LINKAGE. [25], [24], [26]
	ftp://softlib.cs.rice.edu

Table A.1: A table over publicly available linkage analysis tools and where to find them.

GeneHunter	A program to do multipoint linkage analysis. Was written by Leonid Kruglyak, Mark Daly, Mary Pat Reeve-Daly and Eric Lander in 1998. [22][5]
	http://www.fhcrc.org/labs/kruglyak/Downloads/
HOMOG	Programs to analyze heterogeneity (two or more disease loci) with respect to single marker loci or known maps of markers. [32]
	ftp://linkage.rockefeller.edu/software/homog/
HOMOZ	A is a program for rapid multipoint mapping of recessively inheritede disease genes in nuclear families including homozygosity mapping. Was written by Leonid Kruglyak, Mark Daly and Eric Lander in 1995.
	ftp://ftp-genome.wi.mit.edu/distribution/software /homoz
LINKAGE	The core of the LINKAGE package is a series of programs for maximum likelihood estimation of recombnation rates, caculation of LOD score tables, and analysis of genetic risks. [27]
	ftp://linkage.rockefeller.edu/software/linkage/
LINKUTIL	A set of programs useful in linkage analysis. [34]
	ftp://linkage.rockefeller.edu/software/utilities
LIPED	The program carries out genetic linkage analysis, by calculating pedigree likelihoods for various assumed values of the recombination fraction. Can only handle two loci at a time. [31]
	ftp://linkage.rockefeller.edu/software/liped/
Loki	Analyzes quantitative traits observed on large pedigrees using Markov chains, Monte Carlo, multipoint linkage and segregation analysis. Written by Simon C. Heath.
	http://loki.homeunix.net
MAP+	Was written to construct high resolution linkage maps. Requires pairwise sex specific LOD scores, and a trial map containing trial locations for all the loci to be included in the analysis. Written by Dr. A. Collins, J. Teague and Professor N. E. Morton at University of Southampton.
	http://cedar.genetics.soton.ac.uk/pub/PROGRAMS /map+/
MAPMAKER	A software package which performs multipoint linkage analysis. [29]
	ftp://ftp-genome.wi.mit.edu/distribution/software /mapmaker3

Table A.2: Table A.1 continued.

Mendel	Does genetic analysis of human pedigree data under models involving a small number of loci. Written by Professor Kenneth Lange at UCLA.
	http://www.biomath.medsch.ucla.edu/faculty/klange/software.html
MERLIN	Linkage analysis tests for co-segregation of a chromosomal region and a trait of interest. [1]
	http://www.sph.umich.edu/csg/abecasis/Merlin/download/
MultiMap	A program which does automated construction of genetic maps, primary for large-scale linkage mapping. [4]
	http://compgen.rutgers.edu/multimap/multimapdist.html
Pedigree Analysis Package (PAP)	May be used for segregation analysis, variance components analysis, linkage analysis, measured genotype analysis or genetic model fitting. Written by Associate Professor Sandra J. Hasstedt, University of Utah.
	ftp://ftp.genetics.utah.edu/pub/software/pap
SIMULATE	Simulates genotypes in family members for a map of linked markers unlinked to a given affection status locus. [33]
	ftp://linkage.rockefeller.edu/software/simulate/
Superlink	Does multipoint linkage analysis by variable elimination and conditioning of variables by use of Bayesian Networks. [9]
	http://bioinfo.cs.technion.ac.il/superlink
VITESSE	A software package that computes likelihoods. Uses the algorithms of set-recoding and fuzzy inheritance to reduce the number of genotypes needed for exact computation of the likelihood. Written by Associate Professor Dan E. Weeks at the University of Pittsburgh.
	ftp://watson.hgen.pitt.edu/pub/vitesse/

Table A.3: Table A.2 continued.

B Bayesian Networks

This appendix is a short introduction to Bayesian networks and Junction trees, which are applied to genetic analysis in two of the algorithms in this report. It is a short resume of the relevant chapters in [19].

Definition 10 *A Bayesian Network consist of:*

- *A set of variables and a set of directed edges between variables.*
- *Each variable has a finite set of mutually exclusive states.*
- *The variables and the edges form a directed acyclic graph (DAG).*
- *To each variable A with parents $B_1, \ldots, B_n$, there is attached the potential table $\mathcal{P}(A|B_1, \ldots, B_n)$*

A Bayesian network is a Causal network with node set V, where the nodes represent random variables, $X = (X_v)_{v \in V}$, having some joint probability distribution function of the form:

$$f(x) = \prod_{v \in V} f(x_v | x_{pa(v)}) \tag{B.1}$$

with pa(v) denoting the set of parent nodes of the node v. Or said in another way: a Bayesian network is as a graphical representation of the joint probability of all events/variables in the network, see Figure B.1.

The most fundamental property of Bayesian networks are the *d-separation* properties or rules. The d-separation properties are rules of information

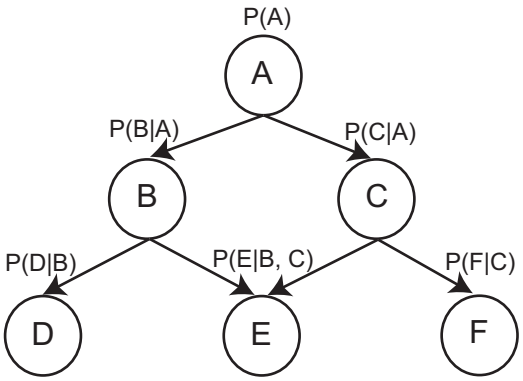


Figure B.1: An example of a Bayesian net.

flow between the nodes of the network. That two nodes are d-separated, means that information given on one node, does not influence our belief in the other, i.e. give us new information which will change the probability distribution of the other node.

The information flow (and blocking of such) between nodes depends on the *connection* in which the node is situated. There are three different types of connections in a Bayesian network: *serial, diverging* and *converging* connection.

Bayesian networks are used for calculating new probabilities when you achieve new particular information. This information is called *evidence*. Evidence is given as binary values, which are multiplied with the original probability tables, such that the impossible values are reset to 0 and the other values are normalized.[1]

In a serial connection, as shown in Figure B.2, A and C are d-separated given *evidence* on B.

In a diverging connection, as shown in Figure B.3, the children of A are d-separated given evidence on A.

In a converging connection, as shown in Figure B.4, the parents variables are d-separated, if there is *no* evidence given on a common descendant, i.e. in this case evidens *opens* the information flow between the parent nodes.

Probability updating in Bayesian networks can be performed by using the chain rule in equation B.2 to calculated $\mathcal{P}(U)$; the joined probability of all the variables in the network.

$$\mathcal{P}(U) = \prod_i \mathcal{P}(A_i|pa(A_i)) \tag{B.2}$$

[1]Sometimes it is easier to wait until after the evidence propagation is completed to normalize the tables.

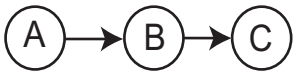


Figure B.2: A serial connection.

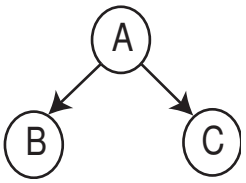


Figure B.3: A diverging connection.

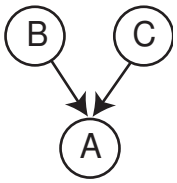


Figure B.4: A converging connection.

where $pa(A_i)$ is the parents of A_i. However the joint probability table increase exponentially in the number of variables, and usually what we really want is to know the effect of the given evidence on a specific variable. The probability updating can be done without ever calculating the full join table, by use of the distributive law which states:

$$\text{If } A \notin dom(\phi_1), \text{ then } \sum_A \phi_1\phi_2 = \phi_1 \sum_A \phi_2$$

where $dom(\phi_1)$ is the domain of potential ϕ_1[2], and $\sum_A$ means that we marginalize A out of the potentials. Marginalization is also called elimination, or if we marginalize A and B out of potential $\phi(A, B, C)$, we say that we *project onto* C, written $\phi^{\downarrow V}$. The order in which we eliminate out the variables also have an effect on the size of the intermediate potentials.

The first step in deciding the *right* elimination order (we will get back to this in a bit) is creating a *domain graph* (also called the *moral graph*) of the Bayesian network. The domain graph is created by removing the directions on the edges of the Bayesian network, and adding edges between parent nodes with common children. These are called moral edges.[3] The domain graph of the Bayesian network in Figure B.1 is shown in Figure B.5.

When we eliminate a variable X we work with the product of all potentials with X in the domain. The domain of this product consist of X and its neighbors in the domain graph, and when X is eliminated, the resulting potential has all of X's neighbors in its domain. In a *perfect elimination sequence* no new domain potentials are created when a variable is eliminated, or said in another way: a bad elimination sequence is an order that result in the need to add extra edges to the domain graph. These edges are called *fill-ins*, and are in Figure B.6 shown as dotted lines. For existing edges the potentials with domain of the connected variables already exist. When adding fill-ins new potentials are created.

Definition 11 *A triangulated graph is an undirected graph with a perfect elimination sequence, meaning no fill-ins are introduced when eliminating variables.*

[2] A potential is a general term for a probability table.

[3] It is called a moral graph because the parents of common children are married.

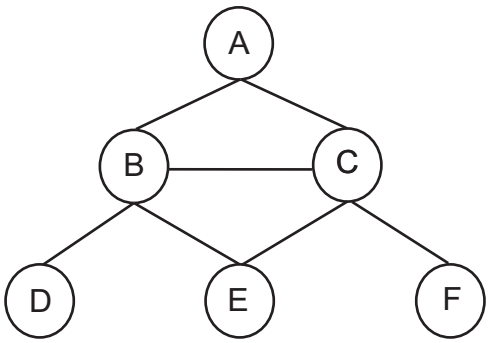


Figure B.5: The domain graph for the Bayesian network in figure B.1.

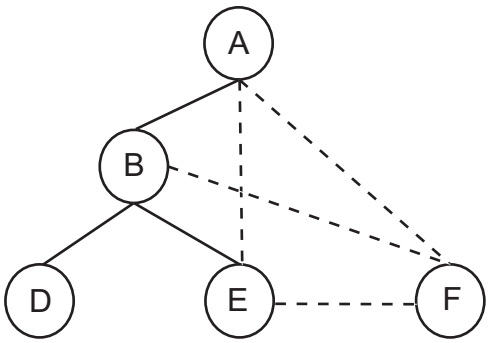


Figure B.6: We want to calculate $P(F)$ and start by eliminating the variable C. This introduces fill-ins between all the neighbors of C, and a probability table over four variables which did not exist before.

If there is a perfect elimination sequence for one variable in a graph, then there is a perfect elimination sequence for each variable in the graph.

Definition 12 *Let G be a set of cliques from an undirected graph and let the cliques be organized in a tree T. T is a join tree if for any pair of nodes v, w all nodes on the path from v to w contain the intersection of v and w.*

If the undirected graph G is triangulated, then the cliques of G can be organized into a join tree.

Definition 13 *A junction tree is a join tree with separators attached to each link. Separators consist of the potentials after variable elimination and two mailboxes; one for each direction in the graph.*

Usually when belief updating a network you project down onto every variable to update with respect to the given evidence. When projection onto several of the variables of a graph, marginalizing out the other variables will often be the same calculations done many times. The benefit of using junction trees is that each variable elimination is done only once, and when calculating $\mathcal{P}$ for each variable in a graph, the results of the variable elimination is only a look up in a mailbox, see Figure B.1.

The big problem is that most graphs are non-triangulated, and our optimized solution using Junction trees is based on triangulated graphs. We need to make our non-triangulated graphs triangulated. This we do by adding fill-ins, but we want to add these in such a fashion that they result in the potentials of the smallest potentials. See Figures B.8 and B.9 for an example of a non-triangulated Bayesian network and its domain graph.

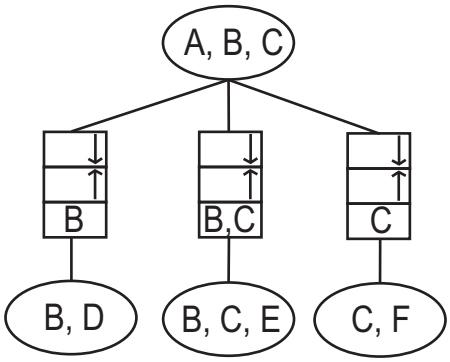


Figure B.7: A junction tree for the Bayesian network in figure B.1.

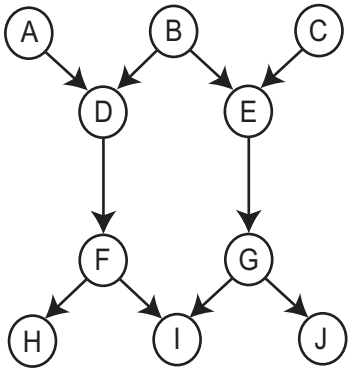


Figure B.8: An example of a Bayesian net and its moral graph. It is clear that the graph is non-triangulated, because it is not possible to eliminate any of the variables B, D, E, F, G without introducing fill-ins.

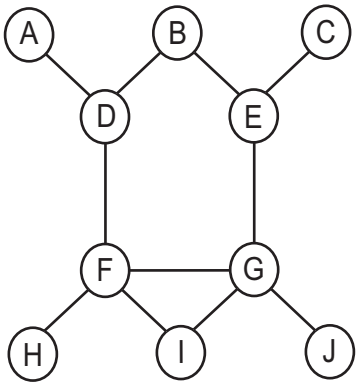


Figure B.9: The moral graph of the non-triangulated Bayesian network in figure B.8.

Adding the optimal fill-ins are unfortunately an NP problem, but the best solution so far is given by the greedy algorithm. The greedy algorithm basically adds one fill-in at a time and adds the fill-in giving the smallest table. See Figures B.12 and B.13.

Another way of propagating evidence in a DAG with multiple paths, is to reduce the DAG to a set of singly connected DAGs. This method is called *conditioning*. Consider the network in Figure B.10, with $\mathcal{P}(A), \mathcal{P}(B|A)$, $\mathcal{P}(C|A), \mathcal{P}(D|B,C)$ We now assume that $A = a$. The DAG is now reduced as shown in Figure B.11 with $\mathcal{P}(B,a), \mathcal{P}(C,a)$ and $\mathcal{P}(D|B,C)$. Now assume that for all states a of A we have a reduced DAG as in Figure B.11. Let evidence e be entered and propagated in all the reduced DAGs, yielding $\mathcal{P}(B,a,e), \mathcal{P}(C,a,e)$, $\mathcal{P}(D,a,e)$ for all a. Then calculate $\mathcal{P}(B,e)$ and $\mathcal{P}(A,e)$. This procedure is called *conditioning on A*.

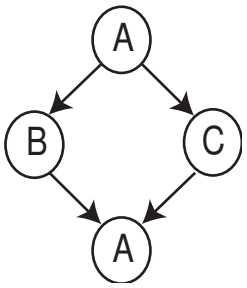


Figure B.10: A Bayesian network with multiple paths.

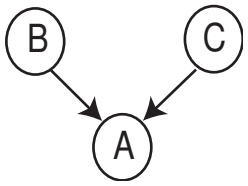


Figure B.11: The reduced DAG when conditioning on A.

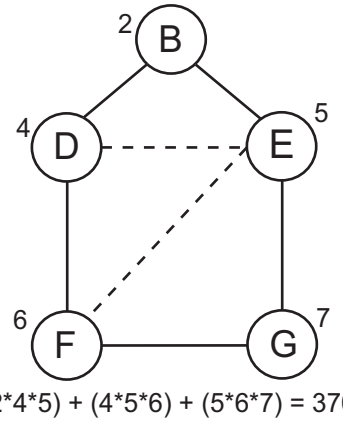


Figure B.12: The dotted lines are the added fill-ins to the graph in Figure B.8 using the greedy algorithm.

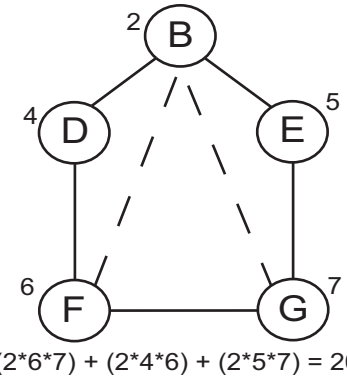


Figure B.13: The dotted lines are optimal solutions to add fill-ins to the graph in Figure B.8.

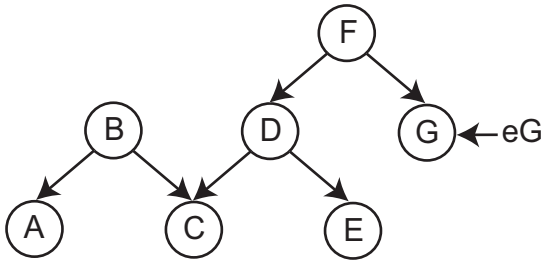


Figure B.14: An example of barren nodes in a bayesian networks. In this example E is a barren node because it does not receive any evidence

The previously described d-separation properties of a Bayesian Network result in another way of reducing calculations. If a node in a Bayesian network has not received evidence and the children of the node has not received evidence, it is called a *barren* node. This is illustrated in Figure B.14, where evidence is given at node G. Because of the d-separation rules nodes

A and B are barren nodes, providing no influence on the calculations of $\mathcal{P}(F)$.

Barren node rule. Let ψ be a set of potentials, and assume that we calculate $\psi^{\downarrow V}$. If $A \notin V$, and the only potential in ψ with A in the domain is of the form $\mathcal{P}(A|W)$, then A is marginalized out by discarding $\mathcal{P}(A|W)$.

C Binary Decision Diagrams

This appendix is an introduction to Binary Decision Diagrams. The purpose of this appendix is to create the basic knowledge needed for understanding the data structure which is used in the current implementation of the linkage analysis algorithm of Allegro, the genetic software package of DeCode.

The following is a short introduction into BDD's which basically is a very short summary over [36]. For more information see [7].

BDDs (Binary Decision Diagrams) can be used as a fast way of determining whether a boolean expression is satisfiable. A boolean expression is satisfiable if it yields true for at least one truth assignment. In general determining whether a boolean expression is satisfiable is NP-complete, but using BDD's reduces this to constant time.

All boolean expressions can be expressed using an *if-then-else* operator $(x \rightarrow y_0, y_1)$ and the constants 0 and 1. The operator is read: if x then y_0 else y_1. If a boolean expression is built entirely from the if-then-else operator and the constants 0 and 1, it is said to be in If-then-else normal form (INF). A boolean expression written in INF can be displayed graphically as a tree, see Figure C.1. The branches corresponding to the *then* part of the operator are called high-branches, and the branches corresponding to the *else* part are called low-branches.

If all equal subexpressions are identified it is no longer a tree of boolean expressions, but a directed acyclic graph called a *binary decision diagram* (BDD), see Figure C.2. The definition of a BDD is:

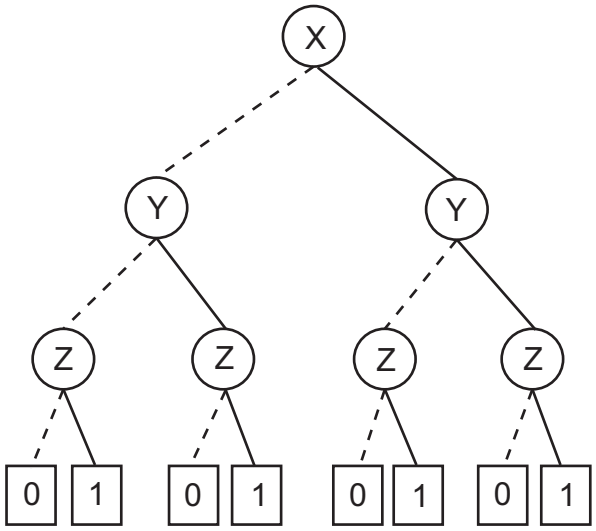


Figure C.1: A decision tree for the expression $(x \wedge y) \vee z$.

Definition 14 *A* binary decision diagram (BDD) *is a rooted, directed acyclic graph with*

- *one or two terminal nodes of out-degree zero labelled* 0 *or* 1*, and*
- *a set of variable nodes* u *of out-degree two. The two outgoing edges are given by two functions* $low(u)$ *and* $high(u)$*. A variable* $var(u)$ *is associated with each variable node.*

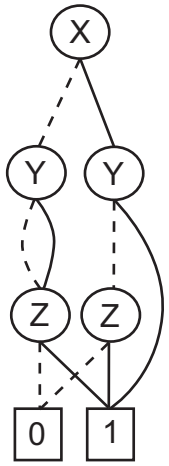


Figure C.2: The BDD of the binary tree in Figure C.1.

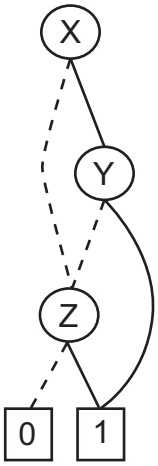


Figure C.3: The reduced version of the graph in Figure C.2.

If the variables occur in the same order on all paths from the root to the leafs of the BDD, it is said to be *ordered*. The size of a BDD is heavily dependent on the order of the variables.

An (O)BDD is said to be *reduced* if

- no two distinct nodes u and v have the same variable name and low- and high-successor, i.e.

$$var(u) = var(v) \wedge low(u) = low(v) \wedge high(u) = high(v) \Rightarrow u = v,$$

- and no variable node u has identical low- and high-successor, i.e.

$$low(u) \neq high(u)$$

When people speak about BDDs they most often mean ROBDDs. Figure C.3 is the reduced version of the BDD in Figure C.2. An important note is that for any boolean function there is exactly one ROBDD which represents it, meaning there is exactly one ROBDD for the constant true or false function.

A Multi Terminal Binary Decision Diagram (MTBDD) is a version of BDD's where instead of two possible terminal nodes of value 0 and 1, there can be several terminal nodes of constant values, see Figure C.4 for an example MTBDD.

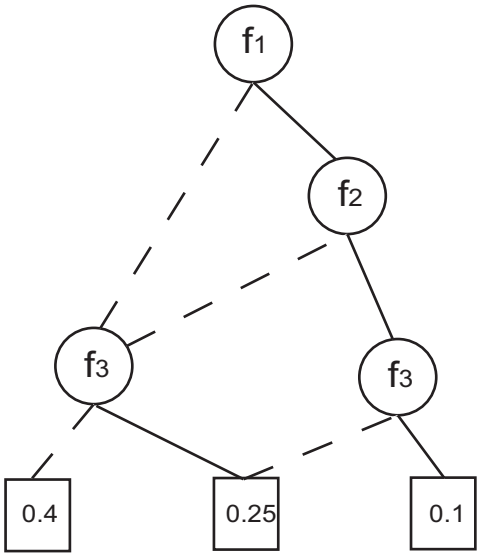


Figure C.4: An example MTBDD.

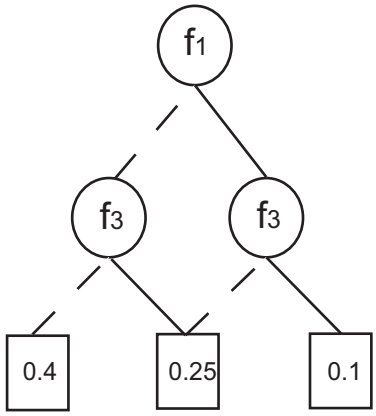


Figure C.5: The diagram in Figure C.4 restricted on the truth value $t^u[1/f_2]$.

Given some truth assignment *restricting* a RO(MT)BDD u is to compute the new ROBDD t^u under the restriction. As an example let u be the MTBDD

shown in Figure C.4. Given the truth assignment $t^u[1/f_2]$ we get the graph in Figure C.5.

D Pedigree Data File

1	1	0	0	3	0	0	1	0	2	4	1	2
1	2	0	0	3	0	0	2	0	2	4	2	3
1	3	1	2	7	5	5	2	0	2	4	1	3
1	4	0	0	7	0	0	1	0	1	4	0	0
1	5	1	2	21	0	0	1	0	1	4	0	0
1	6	0	0	21	0	0	2	0	2	4	1	5
1	7	4	3	26	9	9	2	0	2	3	3	4
1	8	0	0	26	0	0	1	0	1	3	2	2
1	9	4	3	31	11	11	2	0	2	3	0	0
1	10	0	0	31	0	0	1	0	1	3	0	0
1	11	4	3	0	12	12	2	0	2	3	1	5
1	12	4	3	34	14	14	2	0	2	3	1	4
1	13	0	0	34	0	0	1	0	1	3	0	0
1	14	4	3	0	15	15	1	0	1	3	0	0
1	15	4	3	40	17	17	2	0	2	3	0	0
1	16	0	0	40	0	0	1	0	1	3	2	1
1	17	4	3	43	19	19	2	0	2	3	0	0
1	18	0	0	43	0	0	1	0	1	3	3	3

1	19	4	3	0	0	0	2	0	1	3	0	0
1	20	0	0	47	0	0	1	0	0	3	2	5
1	21	5	6	47	22	22	2	0	2	3	0	0
1	22	5	6	48	24	24	2	0	2	3	3	5
1	23	0	0	48	0	0	1	0	1	3	0	0
1	24	5	6	0	25	25	1	0	1	3	2	1
1	25	5	6	0	0	0	1	0	1	3	2	5
1	26	8	7	0	27	27	1	0	1	2	0	0
1	27	8	7	53	29	29	1	0	1	2	3	2
1	28	0	0	53	0	0	2	0	1	2	1	3
1	29	8	7	56	0	0	1	0	1	2	2	4
1	30	0	0	56	0	0	2	0	1	2	0	0
1	31	10	9	0	32	32	1	0	1	2	4	5
1	32	10	9	0	33	33	2	0	1	2	1	5
1	33	10	9	0	0	0	1	0	1	2	1	3
1	34	13	12	0	35	35	2	0	1	2	0	0
1	35	13	12	0	36	36	1	0	1	2	0	0
1	36	13	12	0	37	37	2	0	1	2	1	2
1	37	13	12	0	38	38	2	0	1	2	4	2
1	38	13	12	0	39	39	1	0	1	2	4	2
1	39	13	12	0	0	0	1	0	1	2	1	5
1	40	16	15	0	41	41	1	0	1	2	2	5
1	41	16	15	0	0	0	1	0	1	2	1	2
1	42	0	0	52	0	0	2	0	1	2	0	0
1	43	18	17	52	44	44	1	0	1	2	3	4
1	44	18	17	0	45	45	1	0	1	2	0	0
1	45	18	17	0	46	46	2	0	1	2	0	0
1	46	18	17	0	0	0	2	0	1	2	0	0
1	47	20	21	0	0	0	1	0	1	2	2	2
1	48	23	22	0	50	50	1	0	1	2	3	4
1	49	0	0	51	0	0	2	0	0	2	2	1
1	50	23	22	51	0	0	1	0	1	2	4	5
1	51	50	49	0	0	0	1	0	1	1	2	5
1	52	43	42	0	0	0	1	0	1	1	3	5
1	53	27	28	0	54	54	2	0	1	1	2	3
1	54	27	28	0	55	55	2	0	1	1	1	3
1	55	27	28	0	0	0	1	0	1	1	0	0
1	56	29	30	0	57	57	2	0	1	1	2	5
1	57	29	30	0	0	0	1	0	1	1	4	3

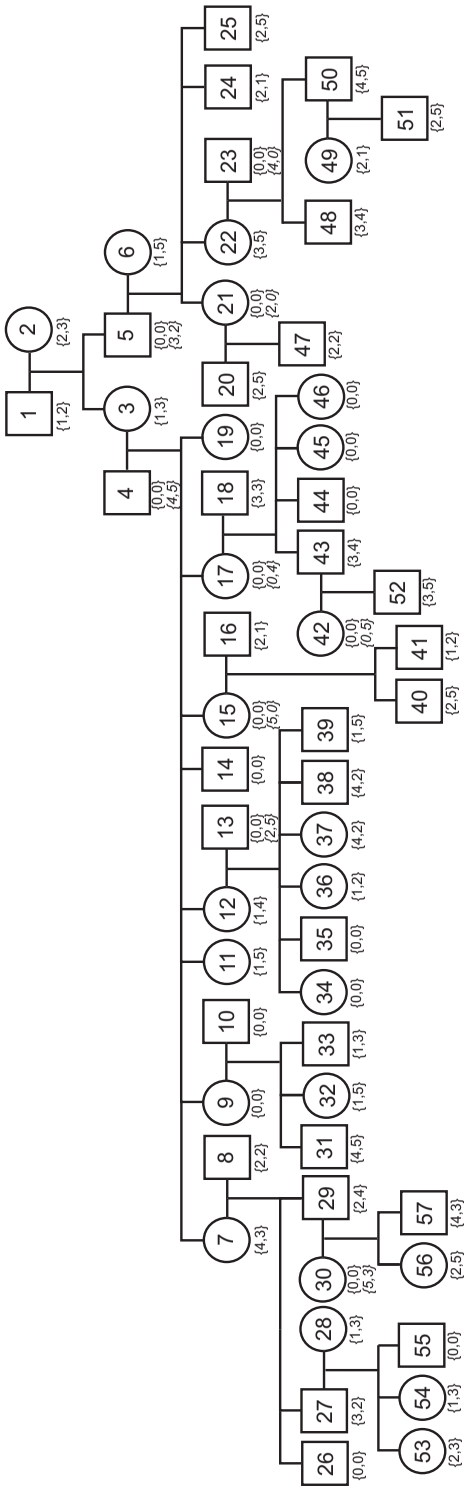


Figure D.1: The pedigree graph given the input data. The genotype of one marker is shown for each individual. Genotypes written in italics are genotype information deduced by doing preprocessing.

Bibliography

[1] G. R. Abecasis, S. S. Cherny, W. O. Cookson, and L. R. Cardon. Merlin - rapid analysis of dence genetic maps using sparse gene flow trees. *Nat. Genet.*, 30:97–101, 2002.

[2] M. Bozga and O. Maler. On the representation of probabilities over structured domains. pages 261–273. CAV'99, Springer, 1999.

[3] Karl W. Broman. Meiosis, recombination and interference. Timestamp: December 5th, 2003.

[4] Aravinda Chakravarti and Tara Cox Matise. The multimap program for construction of linkage maps. Timestamp: Dec. 10th 2003.

[5] Mark J. Daly, Leonid Kruglyak, Stephen Pratt, Nick Houstis, Mary P. Reeve, A. Kirby, and Eric S. Lander. *GENEHUNTER documentation*. Whitehead Institute, MIT, 1998.

[6] David Duffy. Lod score linkage analysis.

[7] Randal E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 1992.

[8] R.C. Elston and J. Stewart. A general model for the genetic analysis of pedigree data. *Human Heredity*, 21:523–542, 1971.

[9] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics Vol. 18*, 2002.

[10] M. Fishelson and D. Geiger. Optimizing exact genetic linkage computations. In *Statistics for Engineering and Information Science*, 2003.

[11] Phil Green. The cri-map program for construction of linkage maps. Timestamp: Dec. 10th 2003.

[12] Daniel Gudbjartsson. Multipoint linkage analysis based on allele sharing models. Technical report, 2000.

[13] Daniel Gudbjartsson, Gunnar Gunnarsson, and Anna Ingólfsdóttir. Bdd-based algorithms in genetic linkage analysis. Technical report, BRICS & deCODE, ?

[14] D. A. Hind and N. Risch. The aspex package: affected sib-pair exclusion mapping, 1996.

[15] Superlink homepage. Timestamp: May 25th, 2004.

[16] Anna Ingólfsdóttir, Anders Lyhne Christensen, Jens Alsted Hansen, Jacob Johnsen, John Knudsen, and Jacob Illum Rasmussen. A formalization of linkage analysis. Technical report, BRICS, 2002.

[17] Anthony J. F. Griffiths, Jeffrey H. Miller, David T. Suzuki, Richard C. Lewontin, and William M. Gelbart. *An Introduction to Genetic Analysis*. W H Freeman, 2000.

[18] M. Jaeger. Probabilistic decision graphs - combining verification and ai techniques for probabilistic inference. To appear in *Int. J. of Uncertainty, Fuzziness and Knowledge-based Systems (special issue with selected articles from PGM-02)*.

[19] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, 2001.

[20] Augustine Kong, Daniel F. Gudbjartsson, Jesus Sainz, Gudrun M. Jonsdottir, Sigurjon A. Gudjonsson, Bjorgvin Richardsson, Sigrun Sigurddardottir, John Barnard, Bjorn Hallbeck, Gisli Masson, Adam Schlien, Stefan T. Palsson, Michael L. Frigge, Thorgeir E. Thorgeirsson, Jeffrey R. Gulcher, and Kari Steffansson. A high-resolution recombination map of the human genome. *Nature Genetics*, 31, 2002.

[21] L. Kruglyak, M.J. Daly, and E.S. Lander. Rapid multipoint linkage analysis of reccessive traits in nuclear families including homozygosity mapping. *Am. J. Hum. Genet.*, 51:519–527, 1995.

[22] L. Kruglyak, M.J. Daly, M.P. Reeve-Daly, and E.S. Lander. Parametric and nonparametric linkage analysis: a unified multipoint approach. *Am. J. Hum. Genet.*, 58:1347–1363.

[23] E. S. Lander and P. Green. Construction of multilocus genetic maps in humans. *Proc. Natl. Acad. Sci.*, 84:2363–2367, 1987.

[24] G. M. Lathrop and J.-M. Lalouel. Easy calculations of lod scores and genetic risks on small computers. *American Journal of Human Genetics*, 36:460–465, 1984.

[25] G. M. Lathrop, J.-M. Lalouel, C Julier, and J. Ott. Strategies for multilocus analysis in humans. *PNAS*, 81:3443–3446, 1984.

[26] G. M. Lathrop, J.-M. Lalouel, and R. L. White. Construction of human genetic linkage maps: Linkelihood calculations for multilocus analysis. *Genetic Epidemiology*, 3:39–52, 1986.

[27] Mark Lathrop and Jurg Ott. *LINKAGE User's Guide*, 1997.

[28] Steffen L. Lauritzen and Nuala A. Sheehan. Graphical models for genetic analyses. Research Report R-02-2020.

[29] E. Lincoln, Stephen, Mark J. Daly, and Eric S. Lander. *MAPMAKER: A Tutorial and Reference Manual*, 1993.

[30] Philip McClean. Lod score method of estimating linkage distances, 1998.

[31] Jurg Ott. *LIPED Computer Program for 2-point linkage*, 1995.

[32] Jurg Ott. *Documentation to Homogeneity Programs*, 1999.

[33] Jurg Ott. *Documentation to the SIMULATE program*, 2002.

[34] Jurg Ott. *Documentation to LINKAGE UTILITY programs*, 2003.

[35] Jurg Ott. *User's Guide to the EH Program*, 2003.

[36] Henrik Reif Andersen. An introduction to binary decision diagrams. Lecture notes, Department of Information Technology, Technical University of Denmark, Lyngby.