RC_Link: Genetic Linkage Analysis using Bayesian Networks ⋆

David Allen *, Adnan Darwiche

Computer Science Department, University of California, Los Angeles, CA 90095, United States

Abstract

Genetic linkage analysis is a statistical method for mapping genes onto chromosomes, and is useful for detecting and predicting diseases. One of its current limitations is the computational complexity of the problems of interest. This research presents methods for mapping genetic linkage problems as Bayesian networks and then addresses novel techniques for making the problems more tractable. The result is a new tool for solving these problems called *RC_Link*, which in many cases is orders of magnitude faster than existing tools.

Key words: Bayesian networks, genetic linkage analysis, pedigree, RC_Link, probabilistic inference

1 INTRODUCTION

Ordering genes on a chromosome and determining the distance between them is useful in predicting and detecting diseases. Detecting where disease genes are located and what other genes are near them on a chromosome can lead to determining which people have a high probability of occurrence, even before symptoms appear, allowing for earlier treatment. Genetic linkage analysis is a statistical method for this mapping of genes onto a chromosome and determining the distance between them (Ott, 1999).

Recently it has been shown that Bayesian networks are well suited for modeling and reasoning about this domain (Fishelson and Geiger, 2002; Allen and

⋆ Available at http://reasoning.cs.ucla.edu/rc_link

* Corresponding author.
Email addresses: dlallen@cs.ucla.edu (David Allen), darwiche@cs.ucla.edu (Adnan Darwiche).

Darwiche, 2003; Lauritzen and Sheehan, 2003; Fishelson et al., 2005). In this paper, we present a new tool called *RC_Link* which can model genetic linkage analysis problems as Bayesian networks and do inference efficiently, in many cases orders of magnitude faster than existing tools.

This paper will first give background on the domain of genetic linkage analysis and Bayesian networks in Section 2 and then in Section 3 will discuss how the linkage problems are encoded as Bayesian networks. Sections 4, 5, and 6 will then present many of the techniques *RC_Link* uses to make the problems more tractable. These techniques can also be used by other existing tools to further improve their performance. Finally, Section 7 will present experimental results comparing *RC_Link* with existing tools and then offer some concluding remarks in Section 8.

2 BACKGROUND

Genetic linkage analysis is an important method for mapping genes onto chromosomes and helping to predict disease occurrences prior to the appearance of symptoms. Research over the past few years has shown that Bayesian networks are well suited for doing linkage analysis computations and many tasks which were considered intractable a few years ago are now solvable, allowing the biology, genetics, and bioinformatics researchers to further study their data and draw new conclusions.

2.1 Genetic Linkage Analysis

Many algorithms used for genetic linkage analysis are extensions of either the Elston-Stewart algorithm (Elston and Stewart, 1971) or the Lander-Green algorithm (Lander and Green, 1987). The first algorithm does well with many people and few genes, while the second algorithm works well for fewer people and many genes. Quite a few genetic linkage analysis tools have been produced, most notably FASTLINK (Cottingham et al., 1993; Schaffer et al., 1994; Becker et al., 1998), GENEHUNTER (Kruglyak et al., 1996), VITESSE (O'Connell and Weeks, 1995), and SUPERLINK (Fishelson and Geiger, 2002). In order to understand the genetic linkage analysis tasks these tools are solving we will first briefly review some relevant Biology background.

Human cells contain 23 pairs of chromosomes, which are sequences of DNA containing the genetic makeup of an individual and are inherited from a person's parents. Each pair consists of one chromosome inherited from the person's father and one from their mother. Locations on these chromosomes are

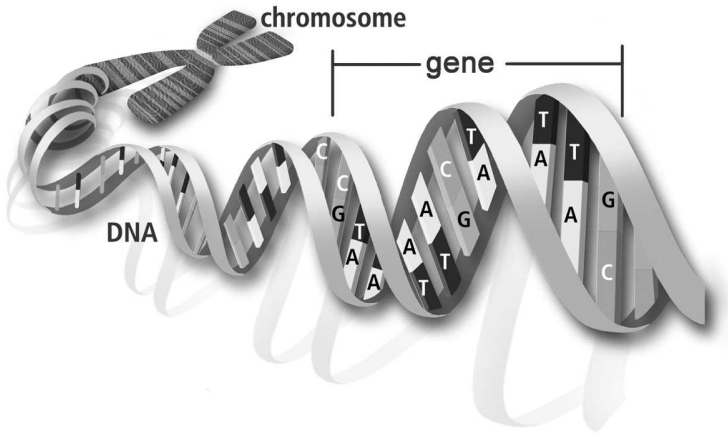


Fig. 1. Diagram of a gene on a chromosome (Courtesy U.S. Department of Energy, Human Genome Program).

referred to as *loci* (singular: *locus*). A locus which has a specific function is known as a *gene*. These functions, which can be a result of a combination of multiple genes, can include such things as determining a person's blood type, hair color, or their susceptibility to a disease. The actual state of the genes is called the *genotype* and the observable outcome of the genotype is called the *phenotype*. A *genetic marker* is a locus with a known DNA sequence which can be found in each person in the general population. These markers are used to help locate disease genes. Figure 1 displays a chromosome, its DNA makeup, and identifies one gene.

Each parent contains their 23 pairs of chromosomes, however they each only pass a total of 23 chromosomes on to their children, one chromosome from each pair, resulting in the child having 23 pairs. It is possible for the transferred copy to be entirely a duplicate of the chromosome from the parent's father or from the parent's mother (the offspring's grandfather or grandmother), however more likely it contains nonoverlapping sequences from both. The locations on the chromosome where the sequences switch between the two parents are known as crossover or recombination events. The *recombination frequency*, θ, (also called the *recombination fraction*) between two consecutive genes is defined as the probability of a recombination event occurring between them. [1] Therefore, if two genes are unlinked, or uncorrelated, they will have $\theta = 0.5$ (meaning the state of the first will not influence the state of the second), whereas linked genes will have $\theta < 0.5$. This frequency is related to the physical distance between them, for example if two genes are close together there may be little chance for a recombination to occur, however if two genes are far away the probability of recombination increases. In Figure 2 a chromosome is depicted along with the location of three (ordered) genes. It furthermore

[1] This frequency between two loci is sometimes measured in units called *centimorgans*, where 1% recombination is equal to 1 centimorgan.

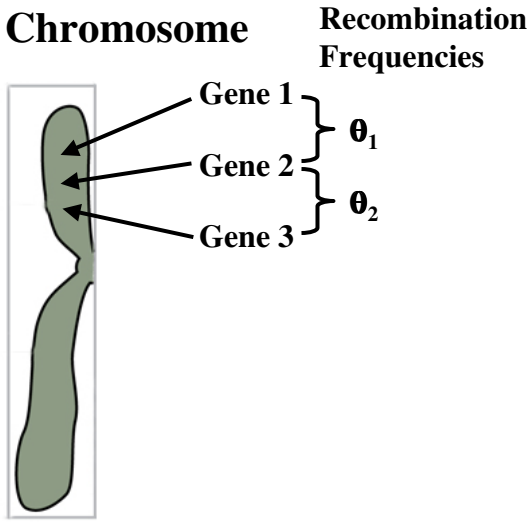


Fig. 2. Ordering genes on a chromosome and determining the distance between them.

depicts the recombination frequencies, θ_1 and θ_2, between the two consecutive pairs.

Therefore, given a large population of people, their inheritance structure (i.e. a family tree, also called a *pedigree*, an example of which is in Figure 3), and partially known genotype and/or phenotype information (e.g. genetic marker readings and disease affection status), genes can be mapped onto the chromosomes based on how frequently recombination events occur between pairs of genes. More formally, let P represent a population of related individuals, let $\mathbf{e}$ be the known evidence on genotypes and/or phenotypes, and let $\hat{\theta}$ be a vector containing the recombination frequencies between each pair of consecutive genes (Hence if we have n genes, then $\hat{\theta}$ will have $n-1$ θ_i values). We can then compute $Pr(\mathbf{e}|P, \hat{\theta})$, which is the likelihood of the known data for the given population and recombination frequencies.

A common task in genetic linkage analysis is then to compute this likelihood for multiple $\hat{\theta}$ vectors, selecting the one with the maximum likelihood. When doing analysis between different populations, the numerical likelihood is generally not too useful, as it is dependent on the size of the population (the larger the population the smaller the likelihood). Therefore, a more useful metric is the LOD score which is ratio of the likelihood to that of the likelihood under 'free' recombination. This score is defined as follows and a score of 3 or more is usually considered significant in determining whether two loci are linked (or correlated).

$$\text{LOD score} = \log[\frac{\text{likelihood with given recombination frequency}}{\text{likelihood with no linkage (a recombination frequency of 0.5)}}] \quad (1)$$

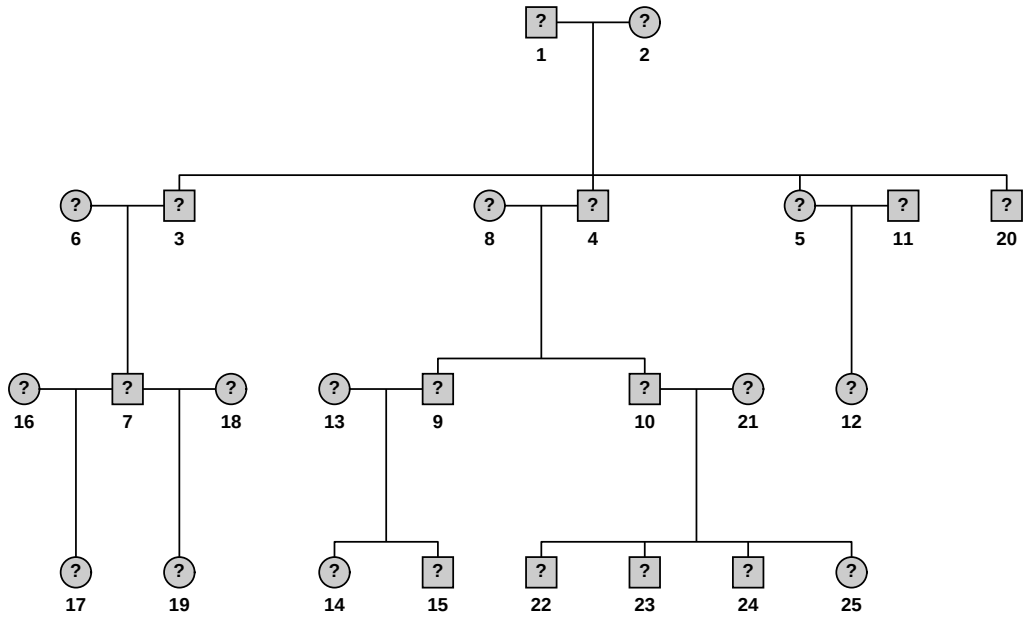


Fig. 3. An example pedigree where squares represent males and circles represent females.

For example, if θ_i is being modified during the maximum likelihood search, then the LOD score would be the log of $Pr(\mathbf{e}|P, \hat{\theta})$ divided by $Pr(\mathbf{e}|P, \hat{\theta}, \theta_i = 0.5)$.

The recombination frequencies with the maximum likelihood (or LOD score) can then be used to determine the distances between the genes. Each different $\hat{\theta}$ vector is a hypothesis of the distance between pairs of consecutive genes, and the goal is to find the one which best fits the known evidence. Therefore the end product of this maximum likelihood search is the most likely distance between each pair of consecutive genes for the known evidence. For a more thorough biological background of genetic linkage analysis see (Ott, 1999).

2.2 Bayesian Networks

Bayesian networks are a method for representing an exponentially large joint probability distribution in a compact manner, while still allowing probability calculations to be done efficiently (Pearl, 1988). They do this by using a graphical structure to model a domain in terms of the independencies between random variables. Bayesian networks are frequently used to probabilistically model situations and to assist in reasoning under uncertainty.

When the genetic linkage analysis domain is modeled as a Bayesian network, the task becomes to compute the probability of evidence, $Pr(e)$, for different recombination fractions (which are used to calculate the LOD score), and then choose the one with the maximum likelihood based on the data. The networks

generated for doing this task frequently are very computationally challenging to do inference on (Allen and Darwiche, 2003). The past few years have seen significant advances in the ability of linkage analysis tools, allowing networks which used to be very challenging to be solved rather easily and networks which were previously too complex that are now tractable.

A Bayesian network is a pair, (G,P), where G is a directed acyclic graph (DAG) and P consists of a set of factors. Each of the nodes in G corresponds to a random variable from a set $\mathbf{X} = \{X_1, \ldots, X_n\}$ and the edges "intuitively" represent direct probabilistic influence between the two connected variables. This influence is measured by the parameters of the network, P. For each $X_i \in \mathbf{X}$, with parents $\mathbf{Pa_i}$, P must contain a factor $f(x_i, \mathbf{pa_i}) = Pr(x_i|\mathbf{pa_i})$. Usually these factors are in the tabular form of a conditional probability table (CPT), where each CPT in P consists of the probability of each state of a variable conditioned on each possible instantiation of its parents. The complete BN compactly defines a joint probability distribution over the random variables.

A graphical depiction of a simple Bayesian network is shown in Figure 4. Each of the random variables has a set of possible states, for example the variable "C: Brain Tumor" can be in the state "Present" or "Absent" and the CPT associated with that variable shows the probability of each, given the state of its parents, which happen to be only the variable "A: Metastatic Cancer." In many BNs, the edges not only depict a probabilistic influence between variables, but also depict causality in the form of cause → effect. Further research in understanding causality in Bayesian networks has also been done (Pearl, 2000).

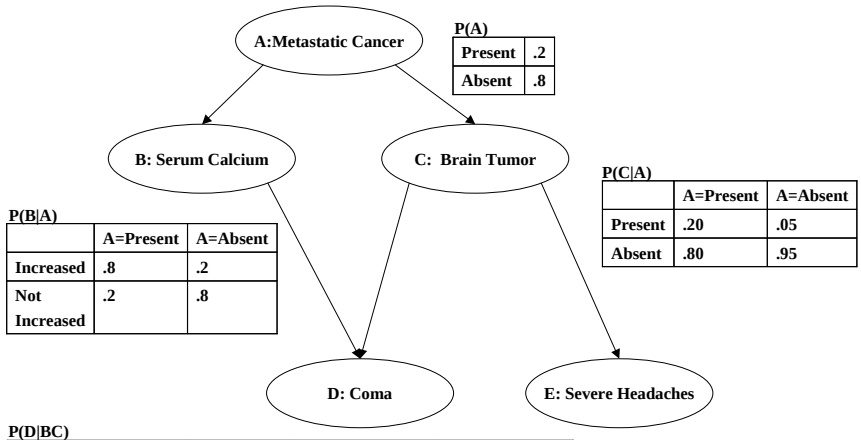


Fig. 4. A Bayesian network.

Once a Bayesian network is created, the task then becomes to do inference on the network (to answer the queries by computing the desired probabilities). Many inference algorithms are based on the initial work of (Pearl, 1986) and (Lauritzen and Spiegelhalter, 1988). Some of the main algorithms in use today are the Hugin jointree algorithm (Jensen et al., 1990), the Shenoy–Shafer jointree algorithm (Shafer and Shenoy, 1990), bucket elimination (variable elimination) (Dechter, 1996), and recursive conditioning (Darwiche, 2001b). Some of these algorithms are compared in (Lepar and Shenoy, 1998), and these and many other algorithms, including many approximation algorithms, were surveyed in (Guo and Hsu, 2002).

It has been shown in the context of Bayesian networks that the Elston-Stewart algorithm and the Lander-Green algorithm can be seen as specific instances of the variable elimination algorithm (Fishelson and Geiger, 2003), where the first eliminates one nuclear family at a time while the second eliminates one gene at a time. The Bayesian network community has many other strategies for generating elimination orders and using tools in this area of research can take advantage of those strategies, as well as other advances in probabilistic modeling.

RC_Link is based on the recursive conditioning algorithm (RC). This algorithm is a divide-and-conquer algorithm, where the problem is represented by the network and is decomposed into smaller problems which can then be solved independently and recursively. This decomposition is accomplished by using conditioning and case analysis, which means fixing the states of a set of variables and then iterating over all possible instantiations (or possible states). This conditioning allows for their outgoing edges to be removed. [2] Therefore, RC picks the conditioning set (called the cutset at each decomposition step) so as to break the network into smaller independent networks, and then recursively solves the smaller networks. An example network decomposition is depicted graphically in Figure 5, where the network corresponding to the root node in the tree is decomposed by conditioning on the variable B. Once B is instantiated the edges between $B \rightarrow C$ and $B \rightarrow E$ are removed, resulting in two independent networks (one containing the variables $\{A, B\}$ and the other containing the variables $\{C, D, \text{and } E\}$). This decomposition structure is known as a decomposition tree (dtree). More formally, a dtree and the variables in the cutset are defined as:

Definition 1 *(Darwiche, 2001b) A dtree for a Bayesian network is a full bi-*

[2] Once a variable is instantiated, its children are no longer dependent on the parent variable, only on the known state and therefore the edge connecting the parent and child can be removed (Darwiche, 2001b).

nary tree, the leaves of which correspond to the network conditional probability tables (CPTs). If a leaf node t corresponds to a CPT ϕ, then $\mathsf{vars}(t)$ *is defined as the variables appearing in CPT ϕ. For an internal node t, with left child t_l and right child t_r,* $\mathsf{vars}(t) \stackrel{def}{=} \mathsf{vars}(t_l) \cup \mathsf{vars}(t_r)$.

Definition 2 *The cutset of internal node t in a dtree is:* $\mathsf{cutset}(t) \stackrel{def}{=} \mathsf{vars}(t_l) \cap \mathsf{vars}(t_r) - \mathsf{acutset}(t)$, *where* $\mathsf{acutset}(t)$ *is the union of cutsets associated with ancestors of node t in the dtree.*

It turns out that many of the subnetworks generated by this decomposition process need to be solved multiple times redundantly, allowing the results to be stored in a cache after the first computation and then subsequently fetched during further computations. Specifically, if we define the context as in Definition 3, then caches can be indexed by instantiations of the variables in the context, as any computation done under the same context instantiation will produce equivalent results.

Definition 3 *The context of node t in a dtree is:* $\mathsf{context}(t) \stackrel{def}{=} \mathsf{vars}(t) \cap \mathsf{acutset}(t)$.

This ability to either cache or recompute computations allows RC to be an any-space algorithm, meaning it can run using any amount of memory. When less than the full amount of memory is used, RC must determine how to best use the available memory (i.e. which computations to store in the caches and which to recompute) (Allen and Darwiche, 2004; Allen et al., 2004; Allen, 2005). Given the above definitions, the pseudocode shown in Algorithms 1 and 2 will compute the probability of evidence, $Pr(e)$ for the input network. A more through description of the algorithm can be found in (Darwiche, 2001b; Allen and Darwiche, 2004; Allen, 2005).

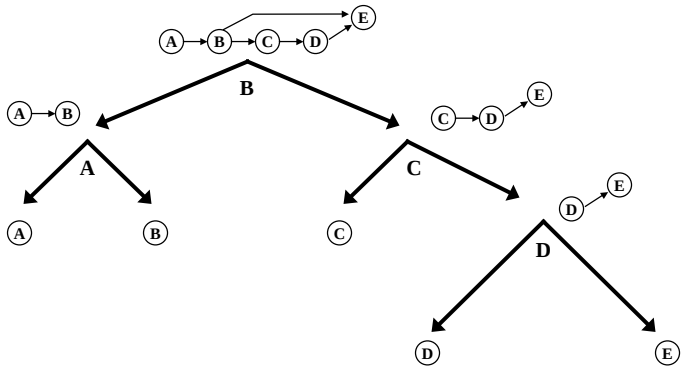


Fig. 5. A decomposition tree (dtree).

Algorithm 1 RC(t): Returns the probability of evidence **e** recorded on the dtree rooted at t

```
1: if t is a leaf node then
2:    return LOOKUP(t)
3: else
4:    y ← recorded instantiation of context(t)
5:    if cache?(t) and cache_t[y] ≠ nil then
6:       return cache_t[y]
7:    else
8:       p ← 0
9:       for instantiations c of uninstantiated vars in cutset(t) do
10:         record instantiation c
11:         p ← p + RC(t_l)RC(t_r)
12:         un–record instantiation c
13:      when cache?(t), cache_t[y] ← p
14:      return p
```

Algorithm 2 LOOKUP(t)

```
φ ← CPT of variable X associated with leaf t
if X is instantiated then
   x ← recorded instantiation of X
   u ← recorded instantiation of X's parents
   return φ(x|u)      // φ(x|u) = Pr(x|u)
else
   return 1
```

3 MODELING LINKAGE ANALYSIS WITH BAYESIAN NETWORKS

The following description is the method both SUPERLINK and *RC_Link* use to map the genetic linkage analysis problem to a Bayesian network (Fishelson and Geiger, 2002) and is currently one of the more common, however this is not the only possible mapping (Lauritzen and Sheehan, 2003). During this discussion we will refer to the simple example in Figure 6. The left side of the figure shows a simple pedigree containing three people (a father, mother, and child). The right side shows the corresponding Bayesian network, assuming three loci are being modeled.

To model the problem as a Bayesian network, for each person in the pedigree and for each locus (genetic marker or gene) being modeled, two random variables will be created called Gp_i and Gm_i, where i refers to the locus. These represent the genotype of the individual (one models the paternal genotype variable and the other is the maternal). [3] For example if the gene we are inter-

[3] Note that many examples in this paper use binary variables for simplicity, however in general the variables are multi-valued.

ested in appears on the first pair of chromosomes, then Gp_i models the value that the gene on the chromosome inherited from the father and Gm_i models the value from the other chromosome in the pair. In addition, a third variable P_i will be created which represents the phenotype, or the observable outcome of the genotype. Edges will be added from each of the two genotype variables to the phenotype variable, as the phenotype is a direct effect of the genotype. This mapping may be deterministic (e.g. the AO blood genotype has the A phenotype) or may be probabilistic (e.g. a person may have the genotype for a disease, but only show symptoms with some probability). For the genetic linkage analysis domain, the input specifies the genotype to phenotype mapping for each locus. Additionally, for those genotype variables associated with founders (i.e. genotype variables which do not have any parents in the network), their prior probabilities are also contained in the input. [4]

For people which are not founders (i.e. people who have parents included in the pedigree) we will create two selector variables for each locus (Sp_i and Sm_i), which determine if the person inherits their parent's paternal genotype or their parent's maternal genotype at that particular locus. One of these selectors, Sp_i, is the paternal selector and therefore edges are added from both of the father's genotype variables and from this selector into the child's paternal genotype variable. The maternal selector, Sm_i, is likewise created along with edges from the mother.

[4] One of the most common input formats is described in the user manual for the LINKAGE tool at http://linkage.rockefeller.edu/soft/linkage/ or also described for the Superlink tool at http://bioinfo.cs.technion.ac.il/superlink/.

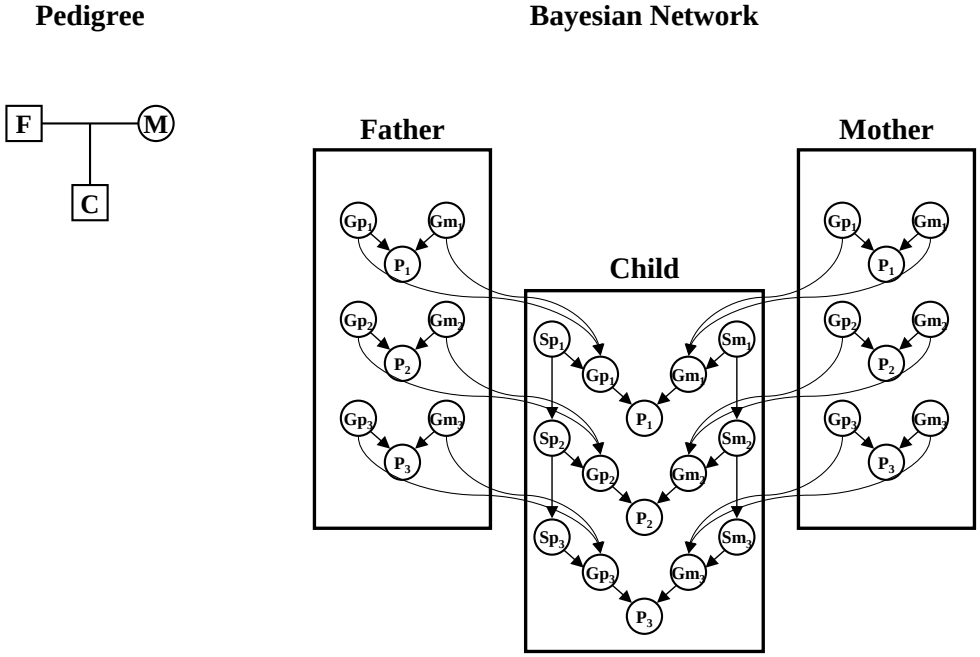


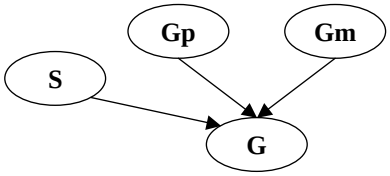
Fig. 6. A pedigree and its corresponding Bayesian network.

An example CPT for a nonfounder genotype variable is seen in Figure 7. From this figure it can be seen that the selector variable simply selects which parent genotype to copy. In the figure, if $S = 1$ the child inherits the parent's paternal genotype value and if $S = 2$ they inherit the maternal.

There are additional edges between consecutive paternal selectors and consecutive maternal selectors. These model the fact that if the preceding locus inherits from either the paternal or maternal haplotype (the paternal haplotype is simply the set of paternal genotype variables, and similarly the maternal haplotype is the set of maternal genotype variables), then the current locus will also inherit from that haplotype, unless a recombination event occurs. Therefore, a recombination occurs when the state of two consecutive selector variables differ.

The selector for the first locus simply has a 50% chance of being in state 1 and similarly a 50% chance of being in state 2. For selectors with parents, their CPTs contain the form shown in Figure 8, where $1-r$ is the corresponding recombination fraction from $\hat{\theta}$. These CPTs intuitively specify that with probability r the process will copy from the same haplotype as the preceding locus and with probability $1 - r$ a recombination event will occur.

After the pedigree is modeled by a Bayesian network, the next step is to compute the likelihood of the known genotype and phenotype evidence for a given pedigree and set of recombination fractions. In Bayesian network terminology, we would create a Bayesian network BN1 based on the pedigree, P, and $\hat{\theta}$, then assert the evidence $\mathbf{e}$, and finally compute the probability of evidence $Pr(\mathbf{e})$. To compute the likelihood for different $\hat{\theta}$ vectors, we can modify the appropriate CPT parameters for each different vector, and for each compute $Pr(\mathbf{e})$.



P(G|S,Gp,Gm)

	S=1				S=2			
	Gp=1		Gp=2		Gp=1		Gp=2	
	Gm=1	Gm=2	Gm=1	Gm=2	Gm=1	Gm=2	Gm=1	Gm=2
G=1	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0
G=2	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0

Interpretation
S=1 implies G=Gp
S=2 implies G=Gm

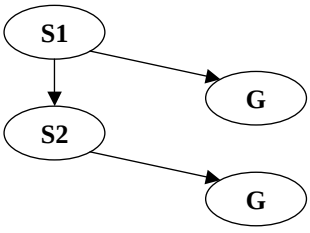
Fig. 7. An example CPT for a nonfounder.

Similarly, we can compute the LOD score by taking the log of ($Pr(\mathbf{e}|BN1)$ divided by $Pr(\mathbf{e}|BN2)$), where BN2 is created from P and $\hat{\theta}$ with $\theta_i = 0.5$.

Currently, many pedigrees of interest create Bayesian networks which are very challenging computationally. They can require significant amounts of memory (sometimes as much as 20 or 30 Gigabytes) using standard algorithms and also can take hours to compute the probabilities (Allen and Darwiche, 2003; Fishelson et al., 2005). These networks are also very large, for example one network with 57 people and 43 loci contained $10,965$ variables. Even more important in determining the computational difficulty is the connectivity of the generated networks.

The Bayesian networks produced can be very challenging, however they also contain a significant amount of determinism. This means that the CPTs contain many zeros and ones (for example the CPTs for nonfounder genotypes deterministically copy one of the two parent genotypes based on the selector, see Figure 7). The implications of this are that when some variables have known values, then other variables can sometimes be determined (or learned). We have previously shown that this can be taken advantage of and that it can lead to significant computational speedups (Allen and Darwiche, 2003; Chavira et al., 2005). This is especially true for probabilistic inference algorithms which work using variable conditioning such as recursive conditioning.

The past few years have shown remarkable improvement on the capabilities of genetic linkage analysis using Bayesian networks. Many of the networks which were previously too costly to do inference on can now be solved using newly developed techniques, however there are still many networks which are challenging (Fishelson et al., 2005). Therefore, exploring additional techniques to further extend the boundary of which networks are computationally feasible is important.



P(S2|S1)

	S1=1	S1=2
S2=1	r	1-r
S2=2	1-r	r

Fig. 8. Format for a selector CPT.

4 PREPROCESSING

This section presents simplification techniques for preprocessing Bayesian networks in order to make inference more tractable while maintaining the ability to compute the correct probabilities. Some of these are used by other probabilistic inference or genetic linkage tools, while others are novel techniques for network simplification. The preprocessing methods are broken down into two groups, those which are applicable to any Bayesian network and those which are dependent on the domain of genetic linkage analysis. Following the simplification technique descriptions, Section 4.3 will discuss the applicability and effectiveness of the different techniques.

4.1 Simplification Techniques for Bayesian Networks

This section presents seven techniques which are useful for any Bayesian network inference algorithm. The first two we present (State Removal and Classical Pruning) are previously existing techniques (Fishelson and Geiger, 2002; Shachter, 1986, 1990). The subsequent five simplifications (Independent Variables, Chain Variables, Single-Child Founder Removal, Irrelevant Edges, and Variable Equivalence) are all new techniques for network simplification.

State Removal

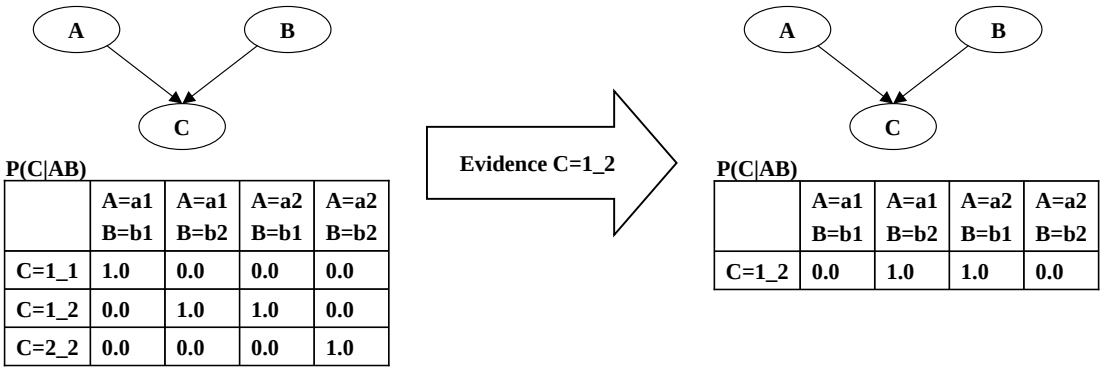
A Bayesian network normally has the property that for all parent instantiations the probabilities of the child states conditioned on the parent instantiation must sum to 1.0. For example, in the CPT on the left side of Figure 9 we note that each column sums to 1.0. Our modeling of the networks presented in the previous section also contains this property, however once the network is generated we will relax this requirement. Specifically, when either the evidence or other simplification techniques determine a state is not possible we will remove it. An example of such a simplification can be seen on the right side of Figure 9, where two states of variable C were removed based on the evidence $C = 1\_2$, leading to CPT columns which do not sum to 1.0. It should be noted that the initial network is a valid Bayesian network, and hence represents a valid joint probability distribution, and that our modified network is probabilistically equivalent to the original network plus the included evidence. The removal of impossible states means the inference algorithm has to look at fewer possible instantiations, allowing them to run faster, although they can then only answer queries with regard to that evidence.

One specific method for detecting states which will always result in a probability of 0.0 and can thereby be removed is value elimination (Fishelson and

Geiger, 2002). An example of this is if X can be in the states $\{x1, x2, x3, x4\}$ and all entries in any CPT containing $X = x1$ contain the probability value of 0 (i.e. $Pr(X = x1, Y, Z) = 0.0$ for all values of Y and Z). In this case, $x1$ can be removed from the domain of variable X, because it is never a valid state. If we examine the left side of Figure 9, but instead of the evidence in the figure let us assume the evidence is $C = 1\_1$. We can then remove the last two rows of the CPT based on our State Removal simplification. Once this is done, if we look at $Pr(A = a2, B, C = 1\_1)$ for every value of B the probability is equal to 0.0. Therefore, we can remove the state $a2$ from the domain of A. Note that we can eliminate states of variables based on any table they are a part of. In this case, we removed a state from variable A, however the table used was the CPT for variable C. The changes will also not only affect the current table, but will change every other table the variable is a part of, as they also will be updated when the states are removed. Specifically, if we remove the state $a1$ from A, then every parent or child instantiation which contains $a1$ will also be removed, possibly leading to further simplifications. In this example, we could additionally remove the state $b2$ from B in a similar fashion to that of $a2$. Since both A and B were binary variables which became unary variables, they now have a known value, which will allow classical pruning to further simplify the network structure.

Classical Pruning

We define classical Bayesian network pruning as removing leaf nodes that are not part of the evidence or query (Shachter, 1986) and removing edges outgoing from observed nodes (Shachter, 1990). We have already seen the removal of edges from observed nodes during the discussion on conditioning. Leaf nodes without evidence can be removed since in any probability calculation on the network that variable's contribution will be a multiplication by 1.0. To see this let us examine the *Lookup*(t) function in Algorithm 2 (Page 9). By definition, variables which only appear in a leaf node do not appear in any cutsets, so when *Lookup* is called the variable will be uninstantiated unless it has evi-



	A=a1 B=b1	A=a1 B=b2	A=a2 B=b1	A=a2 B=b2
C=1_1	1.0	0.0	0.0	0.0
C=1_2	0.0	1.0	1.0	0.0
C=2_2	0.0	0.0	0.0	1.0

	A=a1 B=b1	A=a1 B=b2	A=a2 B=b1	A=a2 B=b2
C=1_2	0.0	1.0	1.0	0.0

Fig. 9. An example of State Removal.

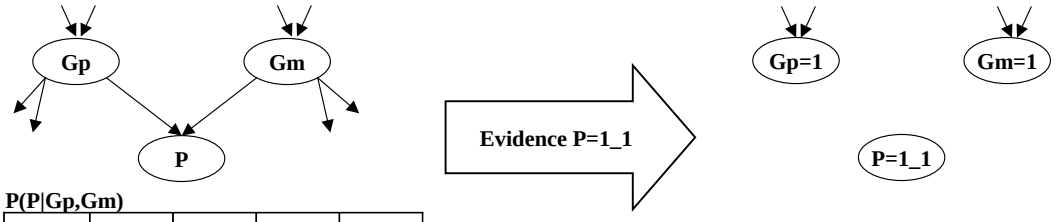
dence. Therefore, for leaf nodes without evidence, *Lookup* will always return 1.0. By examining the remainder of the algorithm it can be seen that the probability value computed will therefore be the same if those leaf variables with no evidence are removed.

Initially we may have many leaf nodes without evidence, for example every phenotype variable is a leaf. Hence, any phenotype variable which does not have evidence associated with it can be pruned. This could also then lead to its parents becoming leaf nodes allowing them to possibly be pruned.

The second rule, removing outgoing edges from observed variables, is not initially applicable. The evidence provided for the genetic linkage domain is usually all on phenotype variables, which are all leaf nodes and therefore don't have outgoing edges. However based on the determinism and other simplifications, additional evidence can be learned and then this rule may be useful and allow for network structure simplification. An example of this can be seen in Figure 10, where we have evidence on the phenotype variable that it is homozygous (i.e. that both genotype variables are equal to one another). The evidence specifies that $P = 1\_1$ and based on the CPT we see that therefore Gp and Gm must both be in state 1. This additional evidence then allows classical pruning to remove outgoing edges from Gp and Gm, in this case removing 6 edges and making P an independent variable.

Independent Variables

In the example in Figure 9, the variables A, B, and C became independent variables, meaning that they don't have any edges coming in or departing from them. Likewise, in Figure 10, variable P became independent. In these cases, if the variables have no evidence associated with them, then since they are leaf nodes they can be removed. However if they do have evidence, it is clear that the lack of edges means that they are independent of the remainder of the network. Therefore they only contribute a constant to any probability



P(P|Gp,Gm)

	Gp=1 Gm=1	Gp=1 Gm=2	Gp=2 Gm=1	Gp=2 Gm=2
P=1_1	1.0	0.0	0.0	0.0
P=1_2	0.0	1.0	1.0	0.0
P=2_2	0.0	0.0	0.0	1.0

Fig. 10. An example of classical pruning and homozygous evidence.

calculation on the network. Note that this evidence could be in the form of standard evidence where we know which state the variable is in, or it could be in the form of states which have been removed (i.e. negative evidence). In general, any independent variable X can be removed from the network and replaced by the constant c, where $c = \sum_{x_i \in X} Pr(x_i)$.

There may be a significant number of these independent variables, depending on the network structure and the simplification techniques used. All these constants can therefore be multiplied together to form a single constant, rather than maintaining each individually. One additional note is that if these variables are query variables or if their network parameters are to be changed, then extra bookkeeping must be used to allow for those to occur prior to the variables' removal.

Chain Variables

Another type of variable which can be preprocessed are those whose indegree is less than or equal to 1 and whose outdegree is equal to 1. We will call these variables *chain variables*, as they participate in a "Chain Structure." This novel technique can be seen in Figure 11. The variables B, C, and E are all chain variables and therefore can be eliminated. To eliminate the variable B, which has parent A and child C, we multiply the CPT for variable B and the CPT for variable C together, resulting in a table over variables A, B, and C. We then sum out the variable B, leaving a table over variables C and A, which will then be the new CPT for variable C. As a result of the multiplication and summation, if we assume all the variables are binary, one example entry in this table for $\{c1|a1\}$ would contain the result of $\{c1|b1\}$ * $\{b1|a1\}$ + $\{c1|b2\}$ * $\{b2|a1\}$.

The second figure shows the network after this is done for B, C, and E (Note that the CPTs for A and D are now different than in the first figure). Looking at this new network we see that the removal of E has now caused A to also become a chain variable and so it can also be eliminated, resulting in the third network. We see that by preprocessing these variables the resulting network has fewer variables, and usually this process will not increase the complexity of inference and it will always maintain the correct computations. We note that by removing the variables we lose the ability to compute marginals over them, however for genetic linkage analysis we are usually only interested in computing the probability of evidence. We also need to be careful if we want to update the network parameters for any chain variable or its child, as once the tables are multiplied together it is harder to determine how to update them. Therefore, variables whose parameters will be changed should either not be eliminated using this rule, or additional bookkeeping must be used to determine how to update the parameters.

Single-Child Founder Removal

Removing chain variables which are roots is a special case of relevance reasoning (Lin and Druzdzel, 1997). Another easily exploitable example of relevance reasoning in genetic networks are pedigree founders (those with no parent information in the pedigree) which only have a single child (note that this refers to the person having a single child, and not to a variable having a single child as we saw for chain variables). See Figure 12 for an example where we have one founder with two loci and a single child (we only show the corresponding genotype and selector variables for the child). These founders contain three variables for each locus: Gm_i, Gp_i, and P_i. The genotype variables each will have a single edge with a common child variable, however they are not chain variables as they also have an edge to the phenotype variable. If the phenotype is unknown, then it will be removed by classical pruning and the genotypes will be removed because they become chain variables. However if the phenotype is known, the previous simplifications will not help. As a group those three variables only contain a single child, and therefore they form a nuisance graph, and can be simplified (Lin and Druzdzel, 1997). One method for simplifying them is to multiply the CPTs for the four variables and then sum out the three variables associated with the founder, leaving a valid CPT for the child variable. We can apply this technique to each locus, therefore the end result will be that all variables associated with the founder will be removed, as seen in Figure 12 where all 6 founder variables have been merged into the child's genotype CPTs.

The notion of relevance reasoning and nuisance graphs are previously known simplifications which are useful for general probabilistic inference. The contribution we make to this is in the realization of how frequently these occur

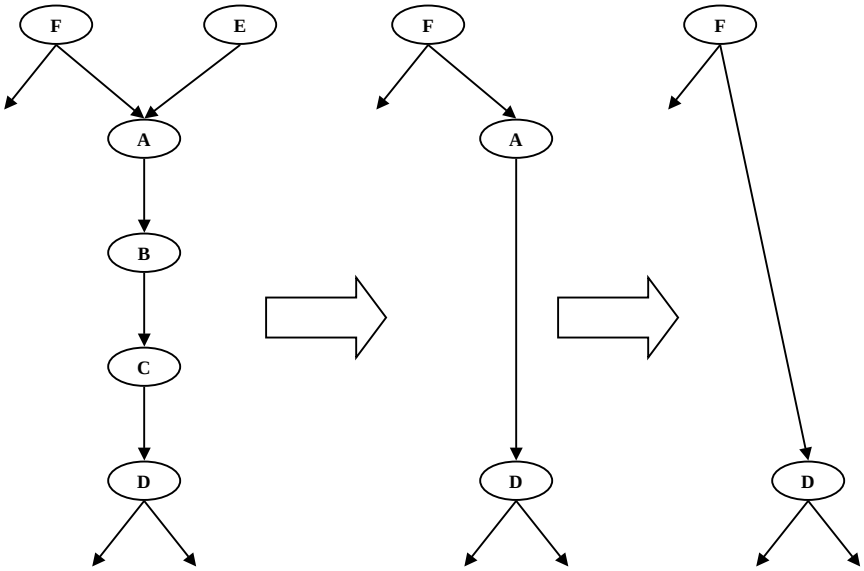


Fig. 11. An example of Chain Variables.

in the genetic linkage analysis domain, and that applying this simplification allows these computations to be done a single time instead multiple times.

Irrelevant Edges

Having an edge between two variables does not guarantee that there is a dependency between them. For example if A is the only parent of B, but $Pr(B)$ is identical for every possible state of A, then the edge between them is irrelevant and can be removed. In our initial genetic linkage model, the above case would never appear, however once the simplifications begin, it is possible for the above to occur. An example of this is given in Figures 13 and 14.

In Figure 13 we initially have 4 binary variables and 2 unary variables (possibly due to knowing evidence on them). One thing we notice is that there are logical constraints between A and B in the CPT for C, and likewise between D and E in F's CPT. Therefore, let us assume we have a simplification which takes advantage of this by combining each of these sets of three variables together (we present such a simplification later called Horizontal Mapping). Once these variables are combined, we end up with a two variable network and the associated CPTs shown at the top of Figure 14 (note that we are breaking our notation and naming the variables with two letters). By then applying value elimination we are able to remove 2 states from each variable, resulting in the middle figure. In the CPT for DE we see the above example of an

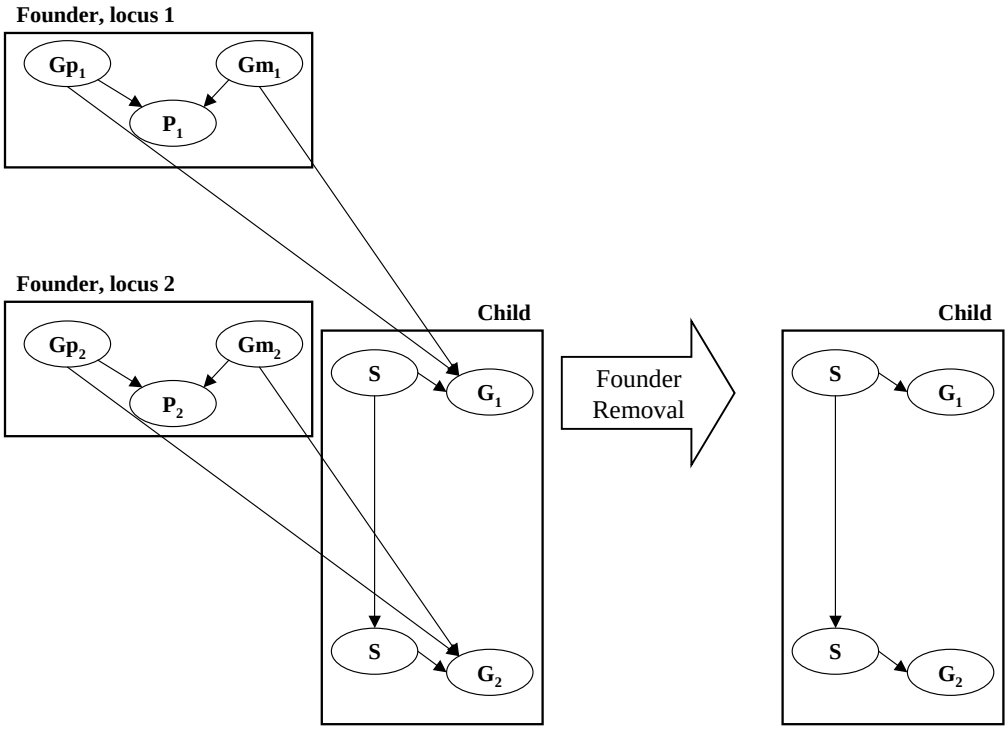


Fig. 12. An example of removing a founder with only one child.

irrelevant edge occur, where no matter what state AB is in, $Pr(d1e1) = r * s$ and $Pr(d2e2) = r * s$. This implies that the edge between AB and DE is unnecessary and can be removed from the network as shown at the bottom of the figure.

In this example we initially had 6 variables which were all connected and based on these simplifications we ended up with only 2 variables which were

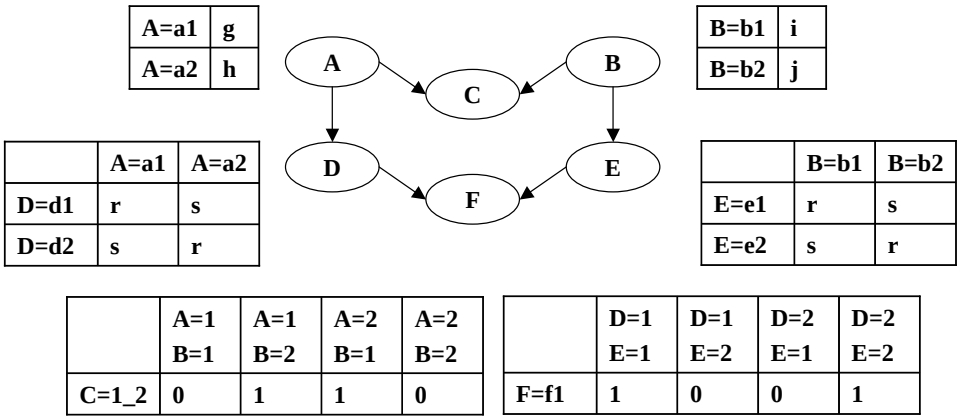


Fig. 13. An example of an irrelevant edge (part 1).

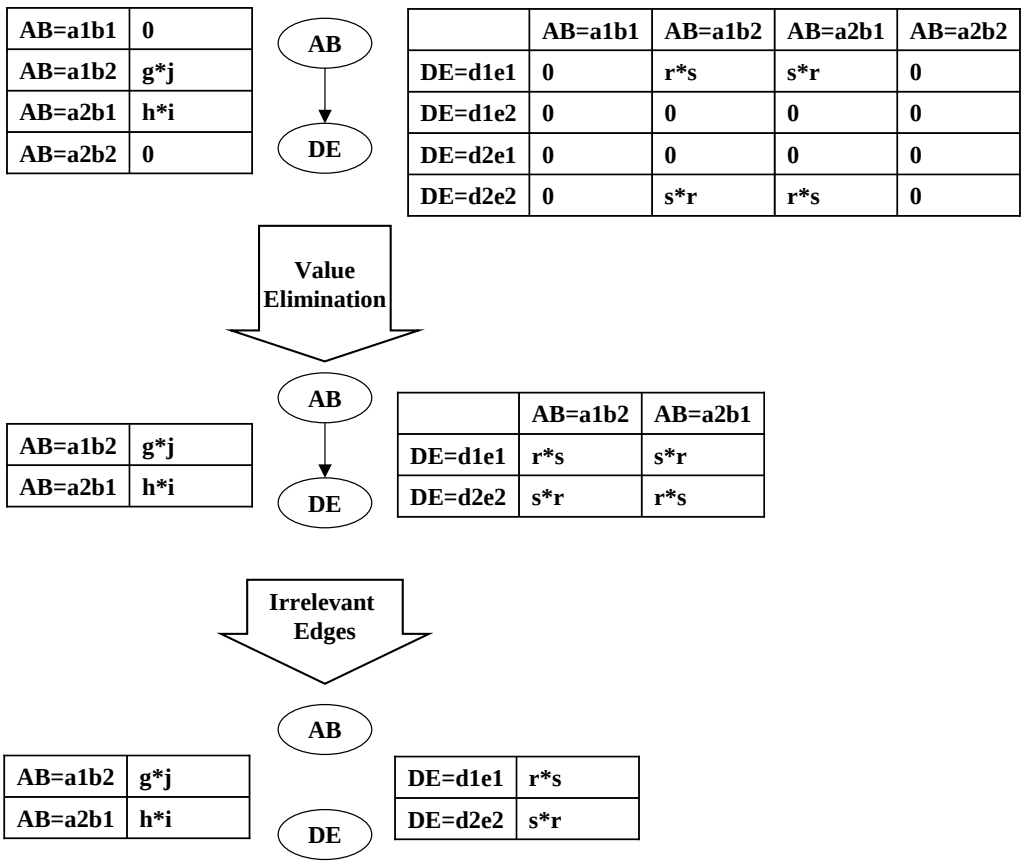


Fig. 14. An example of an irrelevant edge (part 2).

not connected. If D and E initially had other children, we could still do the above simplification, and those children which initially appeared to have dependencies to A and B can be seen to actually be independent of them based on the network parameterization. This novel technique is especially useful in the linkage analysis domain, where many of the CPT parameters contain the same values (e.g. many of the recombination frequencies, or prior probabilities on variables).

This is one example how irrelevant edges could appear in our network as a result of the other simplification techniques, however other combinations of simplifications could also lead to irrelevant edges. This simplification is actually taking advantage of context specific independence (Boutilier et al., 1996), as these irrelevant edges appear when the other techniques simplify the network based on the specific context and as a result two variables become independent.

Variable Equivalence

This novel simplification deals with how to handle variables which are known to be equivalent (We use this term to mean that if either variable is instantiated, then the other will be fixed based on the determinism). For example, if we know the value of a selector, then the child's genotype variable must be equivalent to the corresponding parent's genotype. Let us take a look at a more general example in Figure 15. We are given that the variables A and E must be in the same state.

One method for handling this would be to merge the two equivalent variables into a single variable and remove those states where they were not equal. A graphical example of this can be seen in the middle portion of Figure 15. We initially tried this method in *RC_Link*, however the problem with it is that now representing the new variable is exponential in 6 (the child plus the number of parents), where as previously we had two variables which were exponential in only 3 and 4. To make things worse, the new variable $\{A = E\}$ may also be equivalent to other variables (e.g. if we knew that $A = E, E = H, H = I$), adding even more parents. Eventually the CPT can become either too large to represent, or even if that bound is not reached the larger size will begin to affect the algorithm's performance.

In this section, we propose a novel method for handling variable equivalence, which will not exponentially increase the size of the CPTs, but will still reap the rewards of the variable equivalence. The new method is graphically represented in the lower portion of Figure 15. We create a new variable, named $\{A = E\}$ in the figure, which is a child of both the equivalent variables. [5]

[5] In the more general case of more than two variables, each will contain either 1 or

The new variable's CPT is created such that whenever A and E are equal the probability is 1.0 and otherwise 0.0. We also then move all outgoing edges to one of the equivalent variables. Finally, we will add the new variable and all those variables which do not have the outgoing edges to the beginning of the elimination order (in this example both $\{A = E\}$ and E would be added). The next paragraph will discuss the reason for this modification to the elimination order.

It can easily be seen that these changes will not increase the size of any existing CPTs and the new variable's CPT only contains two parents. However these changes allow two things to occur. First, by adding the variables to the elimination order it forces the algorithm to realize the equivalence earlier in the decomposition, while still allowing the tables to remain in a factored form. Specifically, if we have the three factors $Pr(A, B, C, D)$, $Pr(E, F, G)$, and $Pr(\{A = E\}, A, E)$ and then eliminate the variables $\{A = E\}$ and E the remaining factors will contain $Pr(A, B, C, D)$ and $Pr(A, F, G)$. By contrasting this with the factor for the second graph in Figure 15 which contains $Pr(\{A = E\}, B, C, D, F, G)$ we can see that we still reap the rewards for knowing the variable equivalence (since we have already removed the variable

2 new children, such that all the equivalent variables are connected together.

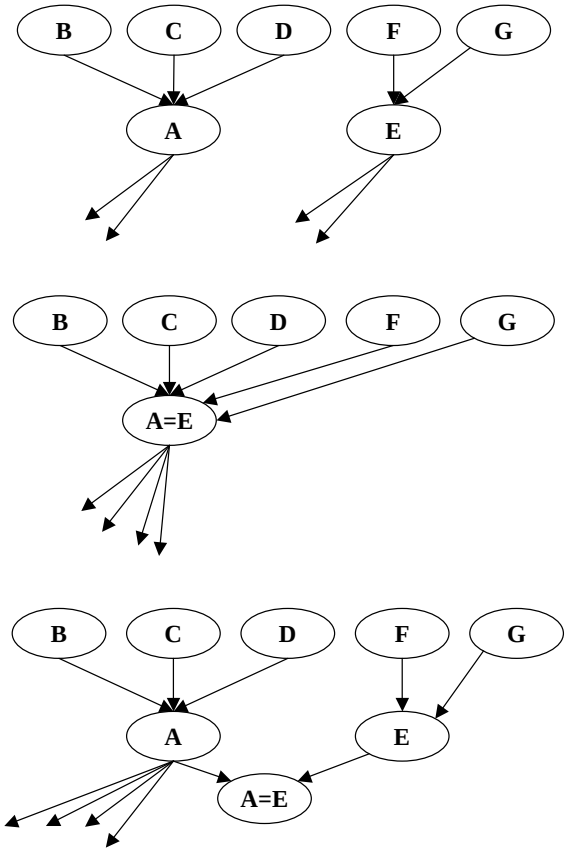


Fig. 15. Two methods for handling variable equivalence.

E), however we don't run into the exponential growth of the CPT (as it is still in a factored form). The second thing the new technique allows is that by moving all the outgoing edges to a single variable, the problem has become easier to decompose, as now we can condition on a single variable A, and remove all children of both A and E, instead of being required to condition on both variables in order to remove the same set of edges. This gives the dtree creation process access to more information about the network in order to enable better decompositions to be found.

This simplification requires outside knowledge that two or more variables are equivalent. This can come from domain specific information (e.g. in genetic linkage, knowing evidence on a selector means a child's genotype variable is equivalent to a parent) or through domain independent methods such as encoding the network's determinism as logical clauses and analyzing them for equivalence. The domain specific knowledge however is not limited to the domain of genetic linkage analysis, hence its inclusion in the section for general Bayesian networks.

4.2 Domain Specific Simplifications

The simplifications in the previous section were general purpose simplification techniques which can be used for any Bayesian network. In this section, we will discuss five simplifications which are specific to the domain of genetic linkage analysis. The first two (Lange and Goradia, and Allele Recoding) are previously known simplification techniques and the third (Phase Removal) is a known technique which we have adapted to Bayesian networks. The final two techniques (Horizontal Mapping and Variable Equivalence for Selectors) are novel methods we have developed.

Lange and Goradia

The Lange and Goradia algorithm (Lange and Goradia, 1987) is used to eliminate states from variables which are never possible. It does this by looking at each nuclear family (i.e. each father and mother and all their common offspring) and each locus separately. For each family, the algorithm attempts to remove states from variables which are never allowed based on the family's variable interactions. For example, if a child state will never occur, no matter which states the parents are in, then that state can be removed. Algorithm 3 contains the pseudocode for it. It should be noted that the algorithm is actually just using logical techniques to remove impossible states, however we classify it as domain specific due to the choice of which variables to examine (i.e. it needs to know which variables are part of each nuclear family, but

Algorithm 3 Lange and Goradia Algorithm

```
1: for each nuclear family do
2:    for each person do
3:       Reduce genotype states based on known phenotypes
4:    for each possible configuration of mother-father genotypes do
5:       Determine all 8 valid children types
6:       if all children have at least 1 valid configuration then
7:          Mark the mother-father states as saved
8:          Mark any child state which is valid for this as saved
9:    for each person do
10:      Remove states not marked as saved
11:   Repeat until nothing new is learned
```

otherwise the techniques involved are general purpose simplifications).

Allele Recoding

Allele recoding was presented in (O'Connell and Weeks, 1995), and was adapted to Bayesian networks in (Fishelson et al., 2005). The basic idea of this simplification is that sometimes it is not possible to distinguish two states for a given variable (an occurrence of this is when no descendants of a person have that state as a known phenotype). More formally, "an allele is defined to be *transmitted* if the following two conditions are fulfilled: (i) the allele appears in the ordered genotype list of a *typed* descendant D of P, as inherited from; (ii) there is some path from P to D containing only *untyped* descendants in the pedigree, namely, D is the nearest typed descendant of P on that path. The remaining alleles are defined to be *non-transmitted*." (Fishelson et al., 2005) For each variable, the non-transmitted states are indistinguishable and can be merged into a single state, taking care to maintain the appropriate inheritance properties (O'Connell and Weeks, 1995). The end result of this simplification is that the variables may have fewer states, and hence are easier to do computations on. We refer the reader to (Fishelson et al., 2005) for the actual implementation details, as they present a simple algorithm for detecting the transmitted states.

Phase Removal

Since founders do not have any information about ancestors, it turns out that their paternal and maternal genotype variables cannot be differentiated. This leads to what is known as phase removal, which detects two equivalent classes of instantiations and removes one of them (Kruglyak et al., 1996). However there are multiple ways of reducing it and the method chosen is significant in determining how beneficial it is. We first define the simplification using an

example and then present a novel method for taking advantage of it in the context of Bayesian networks.

Figure 16 contains a three locus example with a founder (the six genotype variables in the upper left) and three children (only their genotype and selector variables associated with this parent are depicted). We first define a function $InvertPhase(Founder, Child1, Child2...)$, which for any complete instantiation inverts the state of each child's selector variables (i.e. if it was copying the left parent genotype it will copy the right and vice versa) and also it will swap the founder genotypes (i.e. the founder's paternal variable will take on the state the maternal variable had and vice versa).

For any valid instantiation of the variables, the instantiation produced by $InvertPhase()$ will also exist and furthermore both instantiations will be equally likely. This is true since none of the selector variables associated with the child of a founder can contain any evidence (as two genotype variables cannot be distinguished without additional information from ancestors, which founders do not have). Therefore, since no evidence is known on the selectors their inverted state is also possible and equally likely. Likewise, since founder genotype variables cannot be distinguished, any state possible for the paternal genotype must also be possible (and equally likely) for the maternal and vice versa. Therefore, for any instantiation of the variables, the instantiation produced by $InvertPhase()$ will also be valid and will be equally likely.

Based on the fact that for each instantiation there is a second one with equal probability (with a 1 to 1 correspondence), the total number of possible instantiations for the network variables can be reduced by 2^F, where F is the number of founders in the network. Taking advantage of this can lead to significant performance enhancements, however we must first determine how to remove the superfluous phases and update the probability of those remaining.

There are a number of different possible methods for doing this in a Bayesian

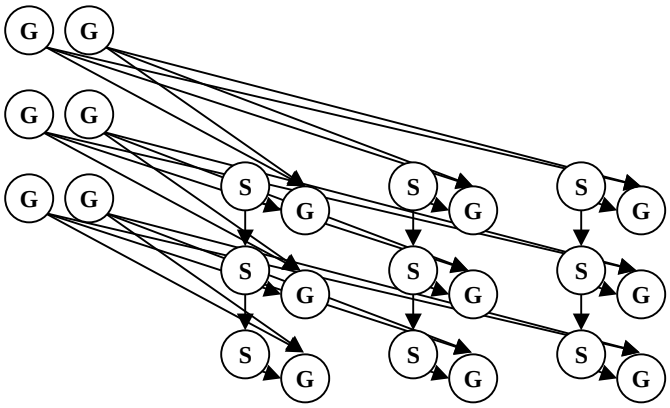


Fig. 16. Phase removal simplification.

network and differing methods will have varying improvements to the inference time. One simple method is to simply pick any single selector associated with each founder and set it to be in one of its states (as both are valid) and then double the probability of that occurrence. It should be apparent that setting the evidence on one of the selectors will then not allow the $InvertPhase()$ function to work (as one of the phases will now be invalidated due to the evidence).

However, based on our experiments with a few additional methods, the one which experimentally seemed to work best was the following. We first pick one of the founder's loci to remove the phase at. We pick this locus by first choosing any which are known to be heterozygous. Whether or not any locus have this property we will then break ties by choosing the locus with the maximum number of child variables with evidence and further break ties by choosing the one with the maximum number of child variables (note that normally each locus would have the same number of children, however due to other simplifications some of the child's variables may have been removed).

Once the locus to simplify is chosen, we will remove the phase there. If the locus is heterozygous (for example it must be either 1_2 or 2_1) then we can merge these variables together using the next simplification (Horizontal Mapping) and then remove one of the two states and double the probability of the other (e.g. remove state 1_2 and double the probability of 2_1). If the founder is not a known heterozygous variable, then we look to see if any of the children have known genotype variables. If so, we set evidence on the corresponding selector and double the probability (e.g. we can set the selector to always copy the left genotype, and since the child's genotype is known, say for example it is in state 1, then all valid parent instantiations must have 1 as the left genotype). Finally, if no children have known genotypes we can still take advantage of the phase removal by again combining the founders genotype variables and then removing the phase there (e.g. for every state i_j where $i \neq j$, remove the state j_i and double the probability of i_j.

Horizontal Mapping

The next domain specific simplification we discuss is a new technique for combining variables, which we call Horizontal Mapping. Our use of this occurs when we know a person's genotype is heterozygous. A heterozygous genotype means that both states are known and different, although it may not be known which state corresponds with which genotype variable (e.g. we may know it is either AO or OA). This contrasts with a homozygous genotype, where both genotype variables are are known and are the same (e.g. AA). With a homozygous genotype, both genotype variables are known and their outgoing edges can then be removed.

Let us assume that we have two genotype variables, call them A and B and that the phenotype variable associated with them is known and the corresponding genotype is heterozygous. Based on this we know that A and B are binary, as each must be in one of the two states stipulated by the phenotype (call the states 1 and 2), and furthermore we know that that $A = 1 \leftrightarrow B = 2$ and $A = 2 \leftrightarrow B = 1$. Together there are four possible configurations of these variables (1_1, 1_2, 2_1, and 2_2), however only two of them are heterozygous. The problem is that with the variables separate, neither variable can have any state removed, as every state is still part of some valid instantiation. Therefore, we combine those variables into a single variable with four states, which then allows us to remove the two non-heterozygous states.

This simplification also enables the decomposition process to realize that setting the state of one of these variables actually removes all the outgoing edges from both (as one variable determines the other and vice versa). Also, since there are only 2 variables, the new CPT will not grow too large, as was seen with the variable equivalence simplification. Therefore, this simplification enables the inference algorithm to skip impossible instantiations (e.g. those which contain 1_1 and 2_2) and also further empowers the decomposition process with additional information.

Variable Equivalence for Selectors

Our final domain specific simplification is a novel technique for determining equivalence between variables in this domain and then advantageously using that to simplify our network. Let us examine Figure 17. We see that the parent genotype is heterozygous (having states 1_2 or 2_1). Likewise we are given 3 children which have their associated selector variable and a known genotype variable.

By using horizontal mapping we will assume that the parent variables are merged into a single variable with two states. Let us examine the case where it is in the state 1_2. From this we see that the first two selectors must copy the 1 from the left genotype variable and the other must copy the 2 from

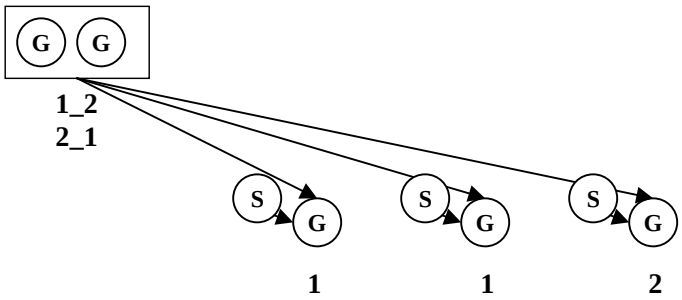


Fig. 17. Selector variable equivalence.

the right. Likewise, if we examine the state 2_1, again all three selectors are determined. In fact, given evidence on any one of the three selectors or the parent determines the states of all the others. Therefore, we see that the selectors are not in fact independent and in fact are very tightly coupled, regardless of the fact that the edge paths between them go through multiple other variables (i.e. the shortest path between them is to go from the selector to the known child genotype, to an unknown parent genotype, back to another known child genotype, and finally to another selector).

This relationship between the selectors and parent genotype variable can help the decomposition process. In order to take advantage of this, we rely on our method for Variable Equivalence in the previous section. Specifically, whenever we have a heterozygous parent, call it X, (e.g. we might know this through evidence on the phenotype) we create an equivalence mapping between X and every selector Y which is associated with a known child genotype variable. As mentioned in the Variable Equivalence section, this empowers the decomposition process with the knowledge about the interrelationship, while still allowing the CPTs to remain in a factored form.

4.3 Simplification Technique Applicability and Effectiveness

Before moving on to the last set of simplifications, those relating to the recursive conditioning algorithm, we will first address the applicability and effectiveness of some of the novel techniques already discussed.

One advantage of these simplification techniques is their reduction of recalculations. Three important places where recalculations are taking place are during each calculation of the probability of evidence, between subsequent queries with different network parameterizations, and when searching for a good network decomposition. By preprocessing our networks with these simplification techniques we can remove some of the recalculations by simply doing computations a single time instead of multiple times during the queries or decomposition search. Some examples of this are the Independent Variables and Chain Variables simplifications. Many standard inference algorithms would readily handle these types of variables, however by simplifying them once during a preprocessing step, we allow our algorithms to not recompute the relevant values multiple times. An example would be the Independent variables, which simply contribute a constant value to the probability query. By preprocessing, we compute this constant once, and then can use it during multiple queries under different parameterizations without recomputing it. Additionally, when we search for a good decomposition, which will be addressed in Section 6, we have already removed these variables simplifying our search. This reduction of recalculations is the main advantage of the Independent Variable and Chain

Variable techniques. The Single-Child Founder Removal technique also assists in reducing recalculations, and in addition can sometimes assist in simplifying the network structure allowing other techniques, such as Chain Variables or Irrelevant Edges to be used. Table 1 shows the size of some networks after the simplification techniques were run and the full size under no simplifications (These networks will be further discussed in the results in Section 7). As can be seen, the preprocessing steps can greatly reduce the size of the network.

The different techniques can have varying levels of effectiveness, and are especially dependent on the amount of evidence known in the network (e.g. genetic marker readings, disease affection status, etc.). Figure 18 depicts the speedup on a few networks from three of the simplification techniques. [6] In the log-plot, the three lines represent the ratio of the time without a given simplification to the time with all simplifications. In these instances, it can be seen that the Phase Removal and Variable Equivalence improved the inference time, however by turning off the Horizontal Mapping (which also turns off the Variable Equivalence technique) we saw the most slowdown. It should be noted here, that when one simplification technique is turned off, others (including those discussed in the next section) may partially compensate for the turned off technique. For example, when variable equivalence is turned off, the decomposition search and skipping computations may still take capture some of the additional instantiations with 0 probability.

In the next two sections we will further discuss simplification techniques which allow us to reuse computations between queries and to search for good network decompositions.

5 OPTIMIZING RECURSIVE CONDITIONING

The last three simplifications are all novel and relate directly to the use of the recursive conditioning algorithm in our genetic linkage computations.

Skipping Computations

The first of the RC based simplifications relates to skipping computations which will have no relevance to the final result. If we examine Algorithm 1 (Page 9), we see on Line 11 that we recursively call RC on each child and multiply the results together. Therefore, if the result on the left child is 0, there is no need to call RC on the right child, as no matter what the result

[6] In order to compensate for the non-deterministic nature of the algorithm, the timings are all averaged over 5 runs.

Table 1
Number of variables in networks after simplifications.

Network	Expected Size	Actual Size
EA7	3570	422
EA8	4590	442
EA9	9435	745
EA10	9690	764
EA11	10965	841
1	752	222
7	2140	423
9	1720	640
13	1680	552
18	1740	619
19	1285	524
20	749	229
23	777	201
25	2155	548
30	2195	629
31	2155	662
33	2315	380
34	2140	484
37	1530	317
38	1020	357
39	2140	449
40	1740	588
41	2155	482
42	763	237
44	1820	483
50	1020	287
51	2300	627

is, it will be multiplied by 0. Hence when 0 is returned from the left child we can skip all the computations on the right. In some cases the result of this can dramatically affect the inference time. We remind you that the number of recursive calls is proportional to the time required to run RC. Table 2 shows the total number of recursive calls (and hence the time requirement) the algorithm would make (labeled as expected), and then displays the actual amount that occurred based on this simplification (labeled Run 1). It also displays the ratio between these two values showing the proportion of actual calls to expected calls. It can be seen that this rule's usefulness varies for each problem, however some of the networks only required 3% of the calls to be made, and every network had as least a small speedup based on this optimization, which requires very little overhead to implement.

Caching Results Between Computations

It is known that when doing multiple computations on the same network that many computations can be reused between different calls. However between each call changes are made in the form of either CPT parameter changes or evidence changes. In classical algorithms it is difficult to determine which computations will become invalid due to these changes, however with RC it is easy. RC simply has to invalidate some corresponding cache values.

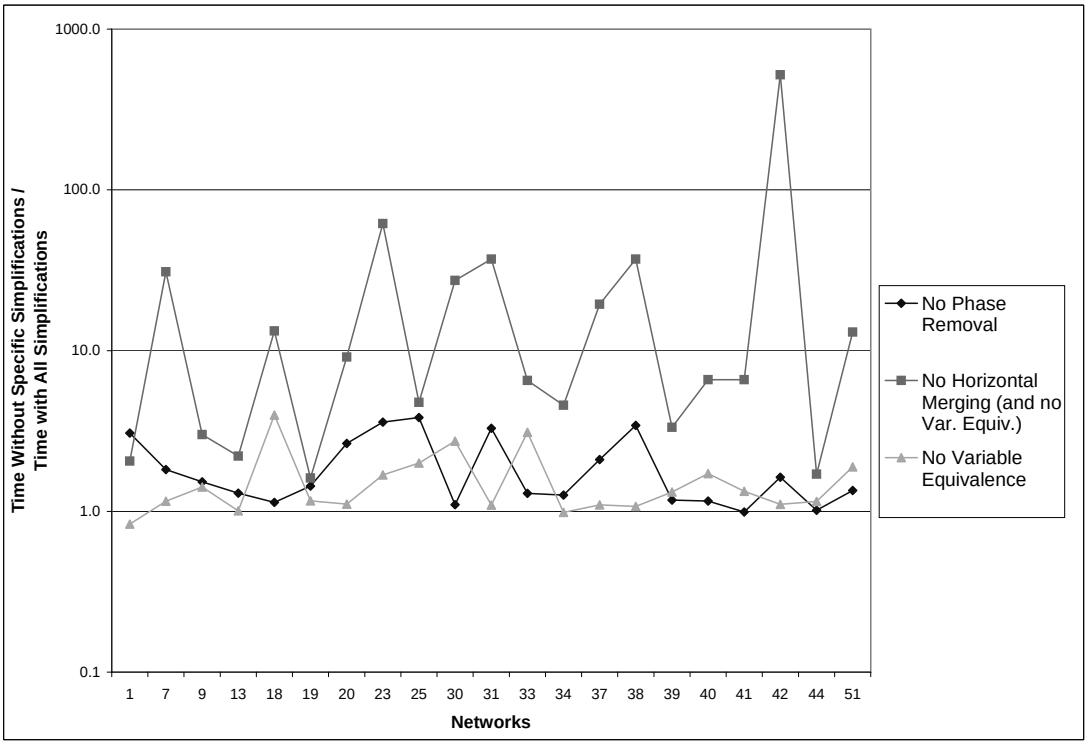


Fig. 18. Simplification technique effectiveness.

Table 2
Experimental results displaying two simplifications (in number of recursive calls).

Network	Expected	Run 1	Run 2	$\frac{Run1}{Expected}$	$\frac{Run2}{Run1}$
1	5.45E+07	5.03E+07	4.73E+07	0.92	0.94
7	3.84E+07	2.80E+07	2.63E+07	0.73	0.94
9	4.46E+08	2.57E+08	2.56E+08	0.58	0.99
13	1.91E+08	1.68E+08	1.62E+08	0.88	0.96
18	9.19E+06	6.93E+06	2.59E+06	0.75	0.37
19	3.76E+07	2.88E+07	2.29E+07	0.77	0.80
20	1.43E+08	6.70E+07	6.59E+07	0.47	0.98
23	5.94E+06	2.29E+06	1.85E+06	0.38	0.81
25	5.89E+07	3.97E+07	3.65E+07	0.68	0.92
30	8.74E+06	6.50E+06	2.50E+06	0.74	0.38
31	2.98E+08	1.89E+08	1.73E+08	0.63	0.91
33	2.30E+06	1.43E+06	3.35E+05	0.62	0.23
34	5.20E+07	4.11E+07	3.92E+07	0.79	0.95
37	7.88E+08	7.96E+07	7.72E+07	0.10	0.97
38	1.25E+10	3.94E+08	3.92E+08	0.03	1.00
39	1.28E+07	8.47E+06	2.40E+06	0.66	0.28
40	9.55E+07	6.92E+07	6.04E+07	0.72	0.87
41	3.44E+07	2.40E+07	3.19E+06	0.70	0.13
42	9.51E+07	5.76E+07	5.25E+07	0.61	0.91
44	1.35E+09	2.19E+08	1.90E+08	0.16	0.87
50	6.17E+12	2.02E+11	2.02E+11	0.03	1.00
51	1.36E+09	6.72E+08	6.44E+08	0.49	0.96

Specifically, given a dtree T, let us consider a change to a CPT parameter in the table associated with the leaf node X. Since RC only ever "passes" information up the dtree, we simply invalidate all cache entries which are located at nodes corresponding to $ancestors(X)$. Similarly, we note that if evidence is changed on a variable whose CPT is located at leaf node Y, we can simply invalidate cache entries corresponding to $ancestors(Y)$. This can be seen since setting the evidence on Y would only affect computations influenced by leaf Y (which we invalidate and would recompute) or computations where Y is a parent to another variable in which case those computations would eventually be multiplied with an ancestor of Y, in which case any computations

which contradict the evidence will be multiplied by a 0.

Table 2 contains the number of recursive calls for Run 1 and also those required during a second run, after changing the recombination frequencies during a maximum likelihood search. It can be seen from comparing the Run 1 column with the Run 2 column how much time was saved due to those saved cache entries. For example, network 44 only required 13% of the calls in order to do the second computation as it required in the first computation. Additionally, for the maximum likelihood search used in genetic linkage analysis, the same parameters are repeatedly changed. Therefore, future runs during the search would require the same number of calls as shown in the Run 2 column.

In the future, it might also be possible to incorporate this information into the decomposition process, as if the changes are known a priori, dtrees may be constructed which specifically allow those changes while minimizing the number of recomputed or invalidated cache entries.

Conditioning with a Knowledge Base

This simplification technique was explored in detail in (Allen and Darwiche, 2003), where it was shown that the combination of a logical knowledge base and conditioning algorithms could sometimes lead to significant timing improvements. This is due to the fact that as each variable is conditioned on, other variables may also be fixed based on the network's determinism. This technique captures many of the same simplifications as the State Removal technique, however it is handled dynamically rather than as a static preprocessing step (as RC conditions on variables in the cutset the technique dynamically learns the value of other variables, leading to simplifications later on in the query). As we exploited additional determinism during the preprocessing phase, less remained in the final network for this technique to take advantage of. Therefore, the current implementation of *RC_Link* has this technique turned off by default, as its overhead versus its benefits can be significant based on the network and its determinism, however is some instances the overhead is still outweighed by the time reduction.

6 DECOMPOSITION SEARCH

The final technique for improving the performance of *RC_Link* deals with finding good network decompositions. For recursive conditioning this means finding a good dtree, or equivalently a good elimination order. SUPERLINK has a sophisticated algorithm for searching for elimination orders (Fishelson et al., 2005). We have capitalized on their algorithm and made two noteworthy

changes to it.

The first deals with the realization that a genetic linkage analysis network can be thought of in terms of a dynamic (or temporal) Bayesian network, where the different loci correspond to the different time points. The GENEHUNTER tool also takes partial advantage of this, in that it treats the network as a Hidden Markov Model (a specific type of dynamic Bayesian network). This however restricts the possible decompositions allowed, where as specialized algorithms for developing elimination orders on dynamic Bayesian networks have been developed (Darwiche, 2001a) and in some cases immediately lead to better orderings.

More formally, a dynamic Bayesian network is a network where each variable X is associated with a time, t. Usually these networks contain the same structure at each time slice and also have the property that edges which go between two different time slices are restricted to going from t to $t+1$ (Darwiche, 2001a). If we examine the Bayesian network in Figure 6 and equate the locus with time, then we notice that the networks within each locus contain the same structure and furthermore the only edges between loci go from selectors at locus t to selectors at $t + 1$. Hence, the method we use to model our genetic linkage networks produces the structural equivalent of a dynamic Bayesian network.

Based on this relationship to dynamic Bayesian networks, the research in (Darwiche, 2001a), and the given pedigree, we developed the following four heuristics for constraining the elimination orders:

- H1: Eliminate all variables at locus t prior to eliminating those at $t + 1$.
- H2: Eliminate all variables at locus $t + 1$ prior to eliminating those at t.
- H3: Set $t = 1$, then eliminate all variables with $locus \leq t$ which do not have temporal edges going to any nodes with $locus > t$. Then increment t and repeat until all variables have been eliminated. (This is based on the notion of a forward interface as defined in (Darwiche, 2001a)).
- H4: Eliminate all variables associated with children in a nuclear family once that family no longer has any descendants which have not already been eliminated. (This can be thought of as removing variables based on where they are located in the pedigree in a bottom up fashion).

When we discuss our final search algorithm we will describe how these four different heuristics for constraining the orderings are used. The H1 and H2 heuristics are very similar to the Lander-Green algorithm (Lander and Green, 1987), which eliminates each locus in order, and the H4 heuristic is based on the Elston-Stewart algorithm (Elston and Stewart, 1971), which does person-by-person elimination.

The second noteworthy addition to the SUPERLINK search is that by generating dtrees instead of elimination orders, we were able to get a more accurate

score metric for measuring the relative quality of the decompositions. With elimination orders, the usual score function is based on the largest cluster size or total state space function (which are both heuristics for jointly measuring both the time and space of the algorithm), where as with a dtree you can compute the actual number of recursive calls, thereby giving you an exact score to compare different dtrees with. When we discuss the actual search below, we also use the dtree structure when picking a variable to eliminate. Since the dtrees determine how much memory is required for the RC algorithm, different candidate dtrees can be compared with regard to both time and space independently. Some dtrees (or elimination orders) produce good initial score values, but turn out to have enormous memory requirements and therefore may not be the most useful decomposition. During the search when we want to quickly compare different dtrees, we assume we can cache at all dtree nodes which require less than a specific threshold [7] and then use the exact number of RC calls as the score. After a dtree is chosen we use the greedy algorithm to find an actual caching scheme based on the available memory on the system (Allen et al., 2004).

We have implemented a search algorithm which combines the SUPERLINK search technique, those based on dynamic Bayesian networks, and the above score function. Algorithm 4 contains the pseudocode for the search. It initially preprocesses and simplifies the network. Then it begins eliminating variables and constructing partial dtrees based on the rules discussed in (Allen and Darwiche, 2002). It starts by creating three dtrees and using the cost of the best as a seed. Then it continues to loop, each time generating a number of dtrees based on probabilistic forms of the min-degree, min-fill, and weighted min-fill heuristics. It dynamically determines when to stop searching by evaluating the time already spent, the quality of the best decomposition (i.e. how much time and space inference would take on it), and the number of consecutive iterations with no improvements.

On Lines 9 to 11 of the algorithm we run the three different heuristics (min-degree, min-fill, and weighted min-fill) each a predetermined number of times, generating a complete dtree for each call to *createDtree*. For each call we randomly determine whether to use one of the four constrained orderings (for example those based on dynamic Bayesian networks) or no additional constraints. Within these bounds we iteratively use the heuristic to propose 3 variables to possibly eliminate. We then probabilistically pick one of these variables based properties of the partial dtree which would be created (e.g. based on the context or number of recursive calls which would be made by this partial dtree).

Table 3 contains some experimental results on the search. The column labeled

[7] In the experiments in the next section this threshold was set to 2GB.

Algorithm 4 Pseudocode for the dtree search algorithm.

```
1: Preprocess the network
2: Eliminate variables based on rules from (Allen and Darwiche, 2002)
3: Run min-degree, min-fill, and weighted min-fill each once and use the best
   dtree as the initial seed
4: Set numTSS=100, numMF=50, and numWMF=50
5: loop
6:     Calculate the expected inference time based on the current best dtree
7:     Stop searching based on the time already spent, the expected inference
       time, and the number of consecutive iterations with no improvement
       (In total attempt to spend approximately 3% to 10% of the expected
       inference time searching)
8:     In cases where an iteration of this loop would take a significant amount
       of time or when the expected inference time is very short, reduce
       numTSS, numMF, and numWMF by either 1/2 or 1/10
9:     Call createDtree(probabilistic min-degree) numTSS times
10:    Call createDtree(probabilistic min-fill) numMF times
11:    Call createDtree(probabilistic weighted min-fill) numWMF times
```

Algorithm 5 Pseudocode for createDtree(elimination heuristic).

```
1: Let H be one of the four elimination order constraints (H1, H2, H3, H4)
   or none with 8%, 8%, 8%, 8%, and 68% probability respectively
2: while some variables are not eliminated do
3:     Use the elimination heuristic passed in to propose 3 variables to possibly
       eliminate next, within the constraints imposed by H
4:     Probabilistically choose one of the 3 proposed variables and eliminate
       it
5: If the new dtree is better than the current best, store it as the current
   best
```

"Initial" is the number of recursive calls based on the initial seed value from the first three dtrees. We then continue to search for better dtrees and display the number of recursive calls for the best dtree, the ratio of the best to the initial, and also the time spent searching. It can be seen on many of the networks, that the search for good decompositions was very useful, in fact on many of the networks it reduced the number of recursive calls by multiple orders of magnitude. For example on network 51 the number of calls was reduced from an exponential of 14 to an exponential of 9, allowing for inference to readily be accomplished on this network.

Table 3
Experimental results displaying dtree search improvement.

Network	Initial	After Search	$\frac{Actual}{Initial}$	Search Time (sec)
1	2.33E+08	5.45E+07	0.23	4.0
7	5.19E+09	3.84E+07	0.01	9.8
9	1.40E+10	4.46E+08	0.03	14.6
13	4.33E+11	1.91E+08	0.00	12.0
18	9.77E+06	9.19E+06	0.94	1.2
19	7.14E+08	3.76E+07	0.05	6.1
20	1.49E+09	1.43E+08	0.10	7.7
23	1.69E+07	5.94E+06	0.35	0.8
25	5.89E+07	5.89E+07	1.00	6.0
30	8.74E+06	8.74E+06	1.00	1.2
31	9.14E+12	2.98E+08	0.00	14.5
33	2.30E+06	2.30E+06	1.00	1.0
34	2.24E+13	5.20E+07	0.00	10.7
37	9.89E+08	7.88E+08	0.80	9.2
38	2.24E+10	1.23E+10	0.55	38.7
39	4.15E+07	1.28E+07	0.31	5.4
40	4.72E+09	9.55E+07	0.02	12.9
41	6.16E+07	3.44E+07	0.56	6.1
42	9.50E+10	9.51E+07	0.00	7.9
44	3.01E+10	1.35E+09	0.04	25.0
50	5.66E+14	1.07E+12	0.00	54.5
51	2.60E+14	1.28E+09	0.00	27.6

7 EXPERIMENTAL RESULTS

In this section we present the overall timing results for *RC_Link* and compare it to SUPERLINK, another state-of-the-art genetic linkage analysis tool, and show that on many networks *RC_Link* is orders of magnitude faster. The following results were performed using *RC_Link* version 2.0 and SUPERLINK version 1.5 (the newest release of both). The experiments were run on an Intel Xeon 2.4GHz processor on a machine with 4GB of RAM (however the currently installed Java virtual machine only allowed *RC_Link* to access 2GB

Table 4
Experimental Results.

			Average (5 runs)(sec.)		Standard Deviation	
Network	# People	# Loci	Superlink	*RC_Link*	Superlink	*RC_Link*
EA7	57	14	1.0	3.9	0.16	0.03
EA8	57	18	4.0	4.3	2.33	0.04
EA9	57	37	8.9	6.4	1.70	0.05
EA10	57	38	15.0	6.7	3.05	0.03
EA11	57	43	16.5	7.2	4.80	0.04
EB3	100	12	9.8	4.9	7.94	0.03
EB4	100	13	4.1	5.1	1.99	0.03
EB5	100	14	5.0	5.4	1.68	0.07
EB6	100	15	9.3	5.6	4.31	0.03
EB7	100	16	10.9	5.8	3.51	0.03
EB8	100	17	9.4	6.2	2.35	0.03
EB9	100	18	9.4	6.3	4.69	0.05
EB10	100	19	9.4	6.5	3.26	0.02
EB11	100	20	12.7	6.7	3.23	0.06

of it.[8]) The results we present in Tables 4 and 5 are from a subset of networks provided with the current and past versions of SUPERLINK.

As both programs use non-deterministic algorithms, their run times may vary significantly from one run to the next. We therefore ran each tool 5 times on each network and provide an average over those runs and also the standard deviation.

It can be seen from the first two datasets (Table 4) that these networks are no longer too challenging for either tool, as both can solve them fairly quickly.[9] We really start to see the differences between the two programs when we analyze the newer, more challenging networks in the third dataset (Table 5). It can be seen that *RC_Link* is faster than SUPERLINK on all these networks except for number 50, and on many of them the difference is orders of magnitude. For example we were able to do network 7 in 23 seconds compared with

[8] Since SUPERLINK is not a Java program, it does not have this limitation.
[9] It should be noted that just a few years ago in 2003 many of these networks were very challenging and some of them were not even able to be solved in a reasonable amount of time (Fishelson and Geiger, 2002; Allen and Darwiche, 2003).

Table 5
Experimental Results.

			Average (5 runs)(sec.)		Standard Deviation	
Network	# People	# Loci	Superlink	*RC_Link*	Superlink	*RC_Link*
1	25	7	78.7	24.5	7.39	0.32
7	25	20	164.6	22.9	2.68	0.36
9	20	20	184.5	168.6	2.15	5.61
13	20	20	234.0	98.7	1.36	2.43
18	20	20	11794.8	5.4	20467.32	0.06
19	15	20	107.2	19.2	6.39	2.45
20	25	7	172.3	34.1	109.21	2.38
23	25	7	216.1	3.2	3.67	0.07
25	25	20	2058.9	26.7	799.25	7.75
30	25	20	2716.3	5.9	973.01	0.14
31	25	20	2064.2	144.9	497.43	12.48
33	25	20	349.6	3.9	92.76	0.06
34	25	20	962.8	31.7	1.83	0.59
37	57	6	1004.8	66.2	481.52	12.03
38	57	4	failed	425.4	failed	298.63
39	25	20	424.5	9.9	0.38	0.48
40	20	20	917.2	57.6	6.05	6.30
41	25	20	665.9	13.4	185.30	5.82
42	25	7	1799.1	36.2	13.79	0.76
44	20	20	2489.9	121.4	7.82	44.59
50	57	4	9712.2	26411.6	4167.19	11127.95
51	25	20	2368.6	556.5	10.72	29.89

165, network 25 in 27 seconds compared with $2,059$, and network 33 in just 4 seconds compared with 350. Even on networks 44 and 51, which SUPERLINK required 2490 seconds and 2369 seconds on, we were able to do significantly faster at just 121 seconds and 557 seconds respectively. Furthermore, it can be seen that *RC_Link* tends to have less fluctuation in its running time, as seen in the smaller standard deviations.

It turns out that our simplification techniques not only significantly speed up the inference algorithm but also drastically reduce the amount of memory

required. For example, the problems in Table 4 which used to require many gigabytes of memory now only require less than 0.2MB each! Additionally, on the more challenging networks in Table 5 we were only required to invoked our time-space tradeoff engine 12 times on 3 different networks (38, 50, and 51). These networks requested up to 5.9GB and since we had to restrict them to 2GB we used our greedy memory allocation algorithm to determine the caching scheme (Allen et al., 2004).

8 CONCLUSIONS

In this paper we introduced the domain of genetic linkage analysis, gave some biological background for the problems, and discussed one method for modeling the linkage problems as Bayesian networks. The main emphasis was on techniques which make the problems more tractable. Specifically we discussed simplifications related to general Bayesian network inference, some domain specific simplifications, and finally some optimizations to the recursive conditioning algorithm. Some of these techniques have already been implemented in other tools, however many of them are novel techniques which could be used to improve the other systems. The techniques which we created or contributed to were: independent variables, chain variables, single-child founder removal, irrelevant edges, variable equivalence, phase removal, horizontal mapping, variable equivalence for selectors, and all those related to the RC algorithm. We finished the paper by comparing our new genetic linkage analysis system, *RC_Link*, to another state-of-the-art system, SUPERLINK, and showed that on many networks our novel techniques allowed it to perform orders of magnitude faster.

References

Allen, D., Darwiche, A., 2002. On the optimality of the min–fill heuristic. Tech. rep., University of California, Los Angeles (UCLA).

Allen, D., Darwiche, A., 2003. New advances in inference by recursive conditioning. In: Uncertainty in Artificial Intelligence: Proceedings of the Nineteenth Conference (UAI-2003). Morgan Kaufmann Publishers, pp. 2–10.

Allen, D., Darwiche, A., 2004. Advances in Bayesian networks. Vol. 146 of Studies in Fuzziness and Soft Computing. Springer–Verlag, New York, Ch. Optimal Time–Space Tradeoff in Probabilistic Inference, pp. 39–55.

Allen, D., Darwiche, A., Park, J. D., 2004. A greedy algorithm for time-space tradeoff in probabilistic inference. In: Proceedings of the Second European Workshop on Probabilistic Graphical Models. pp. 1–8.

Allen, D. L., 2005. Probabilistic inference in bayesian networks using con-

ditioning and their application to genetic linkage analysis. Ph.D. thesis, University of California, Los Angeles.

Becker, A., Geiger, D., Schaffer, A., Jan-Feb 1998. Automatic selection of loop breakers for genetic linkage analysis. Human Heredity 48 (1), 49–60.

Boutilier, C., Friedman, N., Goldszmidt, M. and Koller, D., 1996. Context-Specific Independence in Bayesian Networks. In: Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference (UAI-1996). Morgan Kaufmann Publishers, pp. 115–123.

Chavira, M., Allen, D., Darwiche, A., 2005. Exploiting evidence in probabilistic inference. In: Uncertainty in Artificial Intelligence: Proceedings of the Twenty-First Conference (UAI-2005). AUAI Press, pp. 112–119.

Cottingham, Jr., R., Idury, R., Schaffer, A., Jul 1993. Faster sequential genetic linkage computations. American Journal of Human Genetics 53 (1), 252–63.

Darwiche, A., 2001a. Constant-space reasoning in dynamic bayesian networks. International Journal of Approximate Reasoning 26 (3), 161–178.

Darwiche, A., February 2001b. Recursive conditioning. Artificial Intelligence 126, 5–41.

Dechter, R., 1996. Bucket elimination: A unifying framework for probabilistic inference. In: Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference (UAI-96). Morgan Kaufmann Publishers, pp. 211–219.

Elston, R. C., Stewart, J., 1971. A general model for the genetic analysis of pedigree data. Hum. Hered. 21 (6), 523–542.

Fishelson, M., Dovgolevsky, N., Geiger, D., 2005. Maximum likelihood haplotyping for general pedigrees. Human Heredity 59, 41–60.

Fishelson, M., Geiger, D., 2002. Exact genetic linkage computations for general pedigrees. Bioinformatics 18 (1), 189–198.

Fishelson, M., Geiger, D., 2003. Optimizing exact genetic linkage computations. In: RECOMB'03.

Guo, H., Hsu, W. H., July 2002. A survey of algorithms for real–time bayesian network inference. In: AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems.

Jensen, F. V., Lauritzen, S. L., Olesen, K. G., 1990. Bayesian updating in causal probabilistic networks by local computations. Computational Statistics Quarterly 4, 269–282.

Kruglyak, L., Daly, M., Reeve-Daly, M., Lander, E., Jun 1996. Parametric and nonparametric linkage analysis: a unified multipoint approach. American Journal of Human Genetics 58 (6), 1347–63.

Lander, E., Green, P., 1987. Construction of multilocus genetic maps in humans. Proc. Natl. Acad. Sci. USA 84 (8), 2363–2367.

Lange, K., Goradia, T., 1987. An algorithm for automatic genotype elimination. Am J Hum Genet 40, 250–256.

Lauritzen, S. L., Sheehan, N. A., Nov 2003. Graphical models for genetic analyses. Statistical Science 18 (4), 489–514.

Lauritzen, S. L., Spiegelhalter, D. J., 1988. Local computations with probabilities on graphical structures and their application to expert systems (with

discussion). Journal of the Royal Statistical Society Series B 50, 157–224.

Lepar, V., Shenoy, P. P., 1998. A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions. In: Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference (UAI-98). Morgan Kaufmann Publishers, pp. 328–337.

Lin, Y., Druzdzel, M., 1997. Computational advantages of relevance reasoning in bayesian belief networks. In: Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97). pp. 342–350.

O'Connell, J., Weeks, D., Dec 1995. The vitesse algorithm for rapid exact multilocus linkage analysis via genotype set-recoding and fuzzy inheritance. Nature Genetics 11 (4), 402–8.

Ott, J., 1999. Analysis of Human Genetic Linkage. The Johns Hopkins University Press, Baltimore.

Pearl, J., 1986. Fusion, propagation, and structuring in belief networks. Artificial Intelligence 29, 241–288.

Pearl, J., 1988. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers, San Francisco.

Pearl, J., 2000. Causality: Models, Reasoning, and Inference. Cambridge University Press., Cambridge.

Schaffer, A., Gupta, S., Shriram, K., Cottingham, Jr., R., Jul-Aug 1994. Avoiding recomputation in linkage analysis. Human Heredity 44 (4), 225–37.

Shachter, R. D., 1986. Evaluating influence diagrams. Operations Research 34 (6), 871–882.

Shachter, R. D., 1990. Evidence absorption and propagation through evidence reversals. In: Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence (UAI-90). Elsevier Science Publishing Company, Inc., New York, NY.

Shafer, G. R., Shenoy, P. P., 1990. Probability propagation. Annals of Mathematics and Artificial Intelligence 2, 327–352.