Original Paper

Human Heredity

Hum Hered 610
DOI: 10.1159/0000XXXXX

Received: June 30, 2004
Accepted after revision: December 22, 2004

Maximum Likelihood Haplotyping for General Pedigrees

Ma'ayan Fishelson Nickolay Dovgolevsky Dan Geiger

Computer Science Department Technion, Haifa, Israel

Key Words
Haplotyping · Linkage analysis · Pedigree · SUPERLINK

Abstract
Haplotype data is valuable in mapping disease-susceptibility genes in the study of Mendelian and complex diseases. We present algorithms for inferring a most likely haplotype configuration for general pedigrees, implemented in the newest version of the genetic linkage analysis system SUPERLINK. In SUPERLINK, genetic linkage analysis problems are represented internally using Bayesian networks. The use of Bayesian networks enables efficient maximum likelihood haplotyping for more complex pedigrees than was previously possible. Furthermore, to support efficient haplotyping for larger pedigrees, we have also incorporated a novel algorithm for determining a better elimination order for the variables of the Bayesian network. The presented optimization algorithm also improves likelihood computations. We present experimental results for the new algorithms on a variety of real and semiartificial data sets, and use our software to evaluate MCMC approximations for haplotyping.

Introduction

Haplotype data is valuable in mapping disease-susceptibility genes in the study of Mendelian and complex diseases [Oehlman et al., 1993; Litt et al., 1994]. Such haplotype data defines the nearest flanking recombination events and consequently the smallest interval containing a disease gene. This allows tracing disease genes more easily and cheaply. Other uses of haplotyping include family based statistical tests such as TDT (Transmission Disequilibrium Test) which require haplotype data as input [Ewens and Spielman, 1995], or as a means for detecting genotyping errors, which are usually expressed as an excess of recombination events [Lin and Speed, 1997].

The input data for a haplotyping problem can be divided into two categories: pedigree genotype data and population genotype data. The haplotyping problem is to infer the two haplotypes of each individual from the measured unordered genotypes. Haplotype information from population data is often reconstructed using some evolutionary model and is usually applied to data with a dense map of markers [e.g., Stephens et al., 2001; Gusfield, 2002; Greenspan and Geiger, 2003]. On the other hand, haplotype information from pedigrees is reconstructed using the information that can be inferred on each individual from his relatives' genotypes, and can be used to reconstruct haplotypes from either dense or widely spaced marker data.

The haplotyping problem can be defined via maximizing a suitable likelihood function or via a combinatorial optimization problem. A common combinatorial approach, called the *Minimum Recombinant Haplotype Configuration* (MHRC) problem, is to seek those haplotype configurations that minimize the total number of recombination events observed in the pedigree. Another common combinatorial approach is to seek those haplotype configurations that show no recombination events.

KARGER

Fax +41 61 306 12 34
E-Mail karger@karger.ch
www.karger.com

0001–5652/05/0000–0000$22.00/0

Accessible online at:
www.karger.com/hhe

Dan Geiger
Computer Science Department
Technion, Technion City
Haifa 32000 (Israel)
Tel. +972 4 829 4339, Fax +972 4 829 3900, E-Mail dang@cs.technion.ac.il

Approaches to solve such combinatorial optimization problems include rule-based systems [Wisjman, 1987; Qian and Beckmann, 2002; Li and Jiang, 2003a], graph-theoretic approaches [Gusfield, 2002], dynamic programming [Li and Jiang, 2003b], or linear programming [Li and Jiang, 2004]. These approaches are most appropriate when the expected number of recombination events is small.

The statistical approach for haplotyping by maximizing a suitable likelihood function has been pursued quite extensively [Sobel et al., 1995; Lin and Speed, 1997] and implemented in programs that perform exact computations, such as GENEHUNTER [Kruglyak et al., 1996; Kruglyak and Lander, 1998], ALLEGRO [Gudbjartsson et al., 2000], and MERLIN [Abecasis et al., 2002], as well as in programs that perform approximate computations, such as SIMWALK2 [Sobel and Lange, 1996]. All these methods take into account intermarker recombination fractions or intermarker genetic distances. The objective of these algorithms is to find one or several haplotype configurations of maximum probability given the observed data on the pedigree.

In this paper, we focus on improving exact approaches for generating a maximum likelihood haplotype configuration for larger pedigrees. We present a haplotyping algorithm which we have incorporated into the freely available newest version of SUPERLINK (v1.4), reported herein. SUPERLINK uses Bayesian networks as the internal representation of pedigrees, which allows one to handle a wide variety of linkage problems [Fishelson and Geiger, 2002]. In particular, this representation allowed us to naturally implement a maximum likelihood approach for haplotyping. Furthermore, to support efficient haplotyping on larger pedigrees, we have also incorporated a novel algorithm for determining a better elimination order for the variables of the Bayesian network. This algorithm is especially important when solving linkage problems since the Bayesian networks created for such problems are very large. The presented optimization algorithm also improves LOD score computations. In addition, we have adapted the allele recoding algorithm, presented by O'Connell and Weeks [1995], for the haplotyping task, achieving further reduction in time and space complexity. We present experimental results for the new algorithms on a variety of real and semi-artificial data sets, and use our software to evaluate MCMC approximations for haplotyping via SIMWALK2 [Sobel et al., 1995; Sobel and Lange, 1996].

The Haplotyping Problem

Problem Definition

The sequence of alleles at different loci inherited by an individual from one parent is called a *haplotype*, and the two haplotypes of an individual constitute this individual's *genotype*. A *recombination* is said to have occurred between two loci, if an haplotype of an individual contains two alleles that resided in different haplotypes of the individual's parent. The *recombination fraction* θ is the probability that a recombination occurs between two loci. For a comprehensive background on human genetic linkage analysis consult Ott [1999].

When genotypes are measured by standard measurement procedures, the result is a list of unordered pairs of alleles, one pair for each locus. The *Maximum Likelihood Haplotype Configuration* problem consists of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of the data. The haplotyping problem often does not have a unique solution.

Bayesian Networks

Our model for representing pedigree data is a *Bayesian network*. A Bayesian network is a directed graph with no directed cycles, where each vertex $v = 1, ...; n$ corresponds to a discrete variable X_v and each directed edge represents conditional dependencies between the variables it connects [Pearl, 1988; Lauritzen, 1996]. The distribution of each variable X_v is conditional upon the variables in $\mathbf{Pa_v}$, which is defined as the set of vertices from which there are edges leading into v in the graph. The joint probability of a full assignment $x_1, ..., x_n$ to variables $X_1, ..., X_n$ is the product of these conditional probabilities. In other words,

$$Pr(X_1 = x_1,..., X_n = x_n) = \prod_v Pr(X_v = x_v \,|\, \mathbf{Pa_v} = \mathbf{pa_v}),$$

where $\mathbf{pa_v}$ is the joint assignment $\{x_i | X_i \in \mathbf{Pa_v}\}$ to the variables in $\mathbf{Pa_v}$. From here on, we will use the notation $Pr(\mathbf{y}|\mathbf{z})$ as an abbreviated form of $Pr(\mathbf{Y} = \mathbf{y}|\mathbf{Z} = \mathbf{z})$ for any sets of variables $\mathbf{Y}$ and $\mathbf{Z}$. For example, the joint probability could be rewritten as $Pr(X_1, ..., X_n) = \Pi_v Pr(X_v|\mathbf{Pa_v})$. Note also, that we use capital letters for variable names and lowercase letters to denote specific values taken by those variables. Sets of variables are denoted by boldface capital letters, and assignments of values to the variables in these sets are denoted by boldface lower case letters.

For haplotyping, we consider the *Most Probable Explanation* (MPE) problem for Bayesian networks [e.g.,

Dechter, 1996]. That is, finding an assignment $\mathbf{X} = \mathbf{x^0}$ such that

$$Pr(\mathbf{X} = \mathbf{x^0}, \varepsilon) = \max_{\mathbf{x}} Pr(\mathbf{X} = \mathbf{x}, \varepsilon) = \max_{\mathbf{x}} \prod_{v} Pr(X_v | \mathbf{Pa_v}, \varepsilon),$$

where $\mathbf{X} = \{X_1, ..., X_n\}$ is the set of variables in the Bayesian network, and ε denotes a particular assignment of values to some of the variables in $\mathbf{X}$. The assignment ε is called *evidence*. In the case of haplotyping, the evidence is a partial assignment ε of alleles at some or all loci to people in the pedigree under study.

Another relevant problem for haplotyping is the *Maximum Aposteriori Hypothesis* (MAP) problem [Dechter, 1998], of which MPE is a special case. The input to this problem is the same as for the MPE problem, with the addition of a set of focus variables $\mathbf{A} = \{A_1, ..., A_k\}$, $\mathbf{A} \subseteq \mathbf{X}$, for which the most probable assignment, given the evidence, is desired. The MAP problem is to find an assignment $\mathbf{a^0} = (a_1, ..., a_k)$, such that

$$Pr(\mathbf{A} = \mathbf{a^0}, \varepsilon) = \max_{\mathbf{a}} \sum_{X_j \in \{\mathbf{X} \setminus \mathbf{A}\}} \prod_{v} Pr(X_v | \mathbf{Pa_v}, \varepsilon)$$

Note that when $\mathbf{A} = \mathbf{X}$, MPE is identical to MAP. It has been shown that solving the MAP problem is significantly harder than solving the MPE problem or computing the probability of evidence [Park, 2002]. Consequently, MPE is often solved instead of MAP, and the most likely assignment of all variables is projected on the focus set of variables $\mathbf{A}$.

Methods

Haplotyping in SUPERLINK

Three types of random variables are used in the representation of pedigrees as Bayesian networks in SUPERLINK: *genetic loci* variables which represent the genotypes of the individuals in the pedigree (two genetic loci variables per individual per locus, one for the paternal allele and one for the maternal allele), *phenotype* variables, and *selector* variables which are auxiliary variables used to represent the gene flow in the pedigree. For example, the paternal selector of individual i at locus j indicates whether the paternal allele of individual i at locus j came from his father's paternal haplotype or from his father's maternal haplotype. A similar representation for inheritance within pedigrees has been used previously [e.g., Lander and Green, 1987; Thompson, 1994; Thompson and Heath, 1999]. Figure 1 presents a fragment of a network that describes parents-child interaction in a simple 3-loci analysis. The genetic loci variables of individual i at locus j are denoted by $G_{i,jp}$ and $G_{i,jm}$. Variables $P_{i,j}$, $S_{i,jp}$, and $S_{i,jm}$ denote the phenotype variable, the paternal selector variable and the maternal selector variable of individual i at locus j, respectively. The probability tables that relate these variables are of the following forms:

Transmission Models: $Pr(G_{i,jp}|G_{a,jp}, G_{a,jm}, S_{i,jp})$, $Pr(G_{i,jm}|G_{b,jp}, G_{b,jm}, S_{i,jm})$, where a and b are i's parents in the pedigree. These tables are deterministic, namely, consist solely of zeroes and ones. The first probability table equals 1, if $G_{i,jp} = G_{a,jp}$ and $S_{i,jp} = 0$, or if $G_{i,jp} = G_{a,jm}$ and $S_{i,jp} = 1$. In all other cases, this probability table equals 0. The second probability table is defined analogously.

Population Allele Frequencies: $Pr(G_{i,jp})$, $Pr(G_{i,jm})$ are allele frequencies, where i is a *founder*, namely, an individual in the pedigree whose biological parents are not included in the pedigree. The use of these models is based on the assumptions of Hardy-Weinberg and linkage equilibriums.

Marker Models: $Pr(P_{i,j}|G_{i,jp}, G_{i,jm})$. These tables are also deterministic. The probability table equals 1 if $P_{i,j} = (G_{i,jp}, G_{i,jm})$, or if $P_{i,j} = (G_{i,jm}, G_{i,jp})$. Otherwise it equals 0. The assumption underlying these models is that there are no measurement errors.

Recombination Models: $Pr(S_{i,1p}) = Pr(S_{i,1m}) = 0.5$, $Pr(S_{i,jp}|S_{i,j-1p}, \theta_{j-1})$ and $Pr(S_{i,jm}|S_{i,j-1m}, \theta_{j-1})$, where θ_{j-1} is the recombination fraction between locus $j-1$ and locus j. The recombination fractions between the markers are specified by the user in the input to SUPERLINK. These recombination models do not take genetic interference into account.

Each of these probability tables is called a *factor*. For more details on the structure of the Bayesian network, consult [Fishelson and Geiger, 2002].

We use the following notation to refer to the different variables: $\mathbf{S}$ for the set of all selector variables, $\mathbf{F}$ for the set of genetic loci variables of individuals with no parents in the pedigree (founders), and $\mathbf{N}$ for the set of genetic loci variables of non-founders. For haplotyping, phenotypes are unordered genotypes of typed individuals, and are included in the evidence ε. The Bayesian network of SUPERLINK represents the joint distribution $Pr(\mathbf{S}, \mathbf{F}, \mathbf{N}, \varepsilon)$ in factored form:

$$Pr(\mathbf{S}, \mathbf{N}, \mathbf{F}, \varepsilon) = Pr(\mathbf{S}, \varepsilon) Pr(\mathbf{F}|\varepsilon) Pr(\mathbf{N}|\mathbf{F}, \mathbf{S}, \varepsilon). \quad (1)$$

A maximum-likelihood haplotype configuration of a pedigree is a maximum-likelihood assignment to all the genetic loci variables, namely a joint assignment $\{\mathbf{N} = \mathbf{n_0}, \mathbf{F} = \mathbf{f_0}\}$ which satisfies:

$$Pr(\mathbf{N} = \mathbf{n_0}, \mathbf{F} = \mathbf{f_0}) = \max_{\mathbf{n,f}} \sum_{\mathbf{s}} Pr(\mathbf{s, f, n}, \varepsilon).$$

Since we are interested in determining the most likely gene flow in addition to the most likely assignment to all the haplotypes, we seek a joint maximum-likelihood assignment to the selector variables and the genetic loci variables of founders, namely a joint assignment $\{\mathbf{S} = \mathbf{s_0}, \mathbf{F} = \mathbf{f_0}\}$ which satisfies:

$$Pr(\mathbf{S} = \mathbf{s_0}, \ \mathbf{F} = \mathbf{f_0}) = \max_{\mathbf{s,f}} \sum_{\mathbf{n}} Pr(\mathbf{s, f, n}, \ \varepsilon). \quad (2)$$

The genetic loci variables of non-founders, $\mathbf{N}$, are a function of the genetic loci variables of founders and the selector variables, which, for every $\mathbf{s}$, $\mathbf{f}$, is zero for all values $\mathbf{n}$ except one. Consequently, solving eq. (2) is equivalent to:

$$Pr(\mathbf{S} = \mathbf{s_0}, \mathbf{F} = \mathbf{f_0}, \mathbf{N} = \mathbf{n_0}) = \max_{\mathbf{s,f,n}} Pr(\mathbf{s, f, n}, \ \varepsilon), \quad (3)$$

which is an MPE problem. Thus the MAP problem, defined by eq. (2), is essentially an MPE problem which can be solved more easily.

Our algorithm for solving eq. (3) consists of several stages. The first stage is a preprocessing step of value elimination on the graph

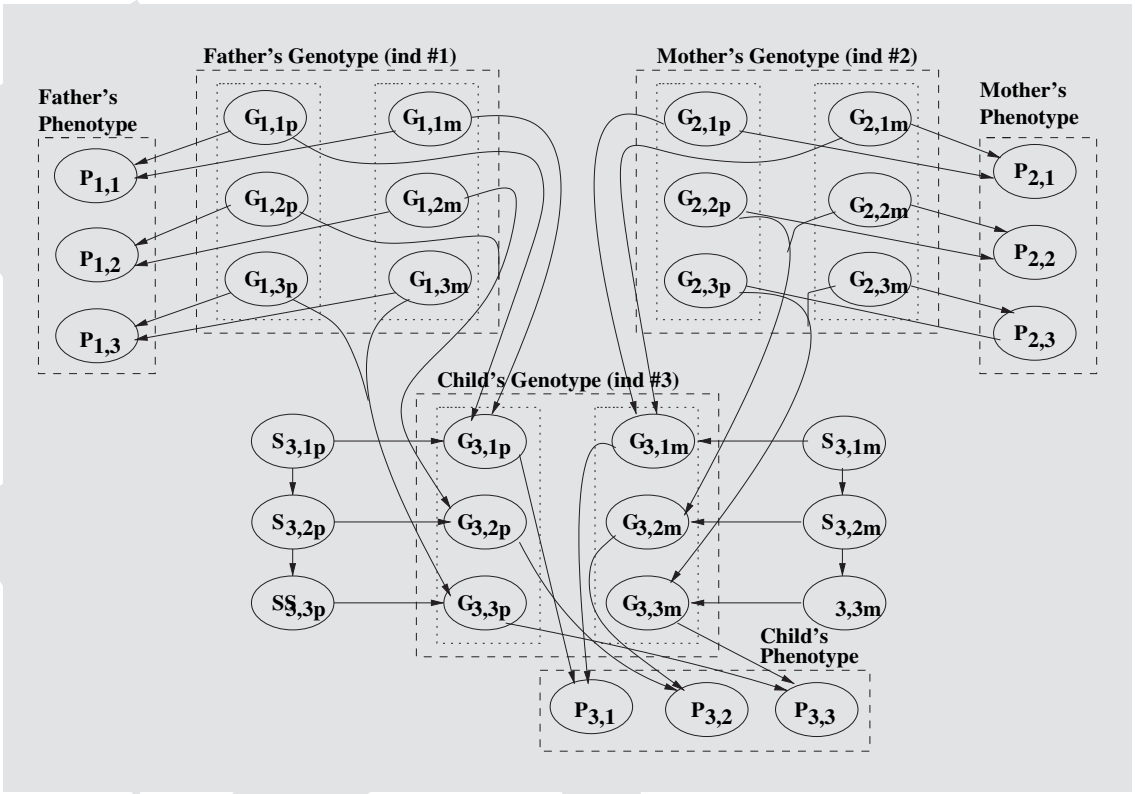

Fig. 1. A fragment of a Bayesian network representation of parents-child interaction in a 3-loci analysis [adapted from Friedman et al., 2000]. The genetic loci variables of individual i at locus j are denoted by $G_{i,jp}$ and $G_{i,jm}$. Variables $P_{i,j}$, $S_{i,jp}$, and $S_{i,jm}$ denote the phenotype variable, the paternal selector variable and the maternal selector variable of individual i at locus j, respectively.

representation of the pedigree [Fishelson and Geiger, 2002]. At this stage inconsistent values for each variable given the evidence are removed, and the values of some of the genetic loci and selector variables can be determined unambiguously from the evidence, namely when all values except one are removed. Mendelian inconsistencies are also discovered at this stage. The value elimination performed is based on the well-known observation that the possible genotypes of an individual can be inferred from the genotypes of his relatives [e.g. Lange and Goradia, 1987]. When the value of a selector variable is determined by value elimination it implies that the parental origin of the corresponding allele is known, i.e., whether it came from the paternal or maternal haplotype of the parent.

After value elimination, we perform allele recoding. In this stage, the genotype lists of untyped individuals are recoded, resulting in a reduction in the number of genotypes that need to be summed over, and hence, in an acceleration of the computations. Our allele recoding algorithm, which is an adaptation of the ideas presented in [O'Connell and Weeks, 1995] to the task of haplotyping, is fully described in the next section.

Finally, haplotyping is done via performing the *Elim-Max* algorithm [Dechter, 1998] on the Bayesian network to determine a maximum-likelihood assignment to the remaining variables. The Elim-Max algorithm, described in the appendix, is a variable elimination algorithm in which variables are eliminated one after another, each time computing the effect of the eliminated variable on the rest of the problem. The order by which variables are eliminated greatly affects both time and space requirements of the computations. In many cases, the memory limitation does not allow solving the problem using variable elimination alone, and hence, variable elimination is combined with conditioning. By conditioning, one means to instantiate some of the variables, perform the rest of the computations for each possible instantiation, and then merge the results. The order of variable elimination and conditioning is determined by a new algorithm described below.

Allele Recoding

When performing likelihood computations or haplotyping, all possible genotype combinations for the individuals in the pedigree need to be iterated over. When using highly polymorphic markers, any person who is untyped at some locus will have a large number of possible genotypes. A possible way to accelerate these computations is to recode alleles and thus reduce the number of possible genotypes that need to be iterated on. Several different methods have been proposed. One method is lumping all alleles that do not appear in the pedigree into a single allele whose population frequency is the sum of frequencies of the lumped alleles [Lange et al., 1988; Schäffer, 1996]. A more efficient method, which recodes the paternal and maternal allele lists of each individual separately, has

Algorithm Allele-Recoding

Input: A pedigree P of size n, where each individual is associated with a list of possible paternal alleles and a list of possible maternal alleles at a given locus l.

Output: Each of the two allele lists of each individual is replaced by a list of sets of alleles.

1. For $i \leftarrow 1$ to n **do** {**Initialize**}

- **If** individual i is typed at locus l, **then**

  Mark all the alleles in both his allele lists as *transmitted*.

  Else

  Mark all the alleles in both his allele lists as *non-transmitted*.

2. Traverse P in a bottom-up manner. Update untyped individual i as follows:{**Mark**}

- **For** each child j of i **do**

  If i is a male, **then**

  Mark each allele of i which appears as transmitted in j's *paternal* allele list as transmitted in i as well.

  Else

  Mark each allele of i which appears as transmitted in j's *maternal* allele list as transmitted in i as well.

- **If** i is an untyped founder, **then** {**Only for haplotyping**}

  Let A_{nt} be the set of non-transmitted alleles of i at locus l, and let $a_k \in A_{nt}$ be the allele with the highest population frequency in A_{nt}.

  Remove all alleles $a_m \in A_{nt}$, $a_m \neq a_k$.

3. For $i \leftarrow 1$ to n **do** {**Recode**}

- Replace each transmitted allele T by the set $\{T\}$.
- Replace all non-transmitted paternal alleles by one set P_n, which consists of these alleles.
- Replace all non-transmitted maternal alleles by one set M_n, which consists of these alleles.

4. Return.

Fig. 2. The allele recoding algorithm for the haplotyping problem. For likelihood computations, the second stage of step 2 is removed.

been suggested by O'Connell and Weeks [1995], and implemented in VITESSE. The allele recoding algorithm implemented in SUPERLINK is based on the ideas of *set-recoding* and *fuzzy inheritance* defined in VITESSE. These definitions are repeated here for completeness. Our contribution is the adaptation of this algorithm to the task of maximum likelihood haplotyping for general pedigrees.

The allele-recoding algorithm is based on the observations that alleles have two roles in likelihood computations, and that valid recoding does not alter these roles:

1. Determine prior probabilities of founders' genotypes. The genotype frequency of a founder is computed using the population frequencies of the two alleles that constitute the genotype, assuming Hardy-Weinberg equilibrium.

2. Determine recombination events. A recombination event is determined by identifying the parental origin of the child's alleles, namely, whether the child's alleles came from the paternal or maternal haplotype of his parent. Note that the allele identity does not matter here; only whether the allele matches the parent's paternal or maternal allele.

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*. In terms of determining recombination events, a person's non-transmitted alleles are indistinguishable from one another by data, and can therefore be combined into a single representative allele.

The allele recoding algorithm (fig. 2) is executed after initial value elimination is performed on the input pedigree as described in the previous section. At this stage, each individual is associated with two allele lists at each locus, a paternal allele list and a maternal allele list. In the first stage of the allele recoding algorithm, all alleles of typed individuals are marked as transmitted, and all alleles of untyped individuals are marked as non-transmitted. Next, the pedigree is traversed in a bottom-up manner where each untyped person is updated after his children have been updated. The update is performed as follows: each allele of the father which appears as transmitted in the paternal allele list of the child is marked as transmitted in the father as well. A similar update is done for mothers. After an untyped founder has been updated by all his children, a final processing is performed on the founder's two non-transmitted allele lists. In this stage, only the highest frequency allele in each of the two lists is kept. At the end of the algorithm, each set of non-transmitted alleles forms a set (e.g., $\{A, B, C\}$), and each transmitted allele A forms a set including only itself, i.e., $\{A\}$. Recall that, if a parent has the ordered genotype $A|B$ and his child has allele C, then C is inherited from the parent if $A = C$ or $B = C$. After allele recoding, however, A, B, and C are now sets of alleles, and hence C is inherited from the parent if $A \subseteq C$ or $B \subseteq C$. This is termed *fuzzy inheritance* in [O'Connell and Weeks, 1995].

In the appendix, we prove that the probability of the assignment found for the regular case (without allele recoding) is the same as the one found in the case of allele recoding. This claim proves the correctness of the allele recoding algorithm for maximum likelihood haplotyping. Note that there is often more than one maximum likelihood assignment, but the algorithm described herein produces only one. To produce all possible maximum likelihood assignments, one needs to change the Elim-Max algorithm, described in the appendix, to store all optimizing values of each variable X_v rather than storing a single optimizing value. This addition increases the time and space complexity of the computations.

Computation Order

The problem of determining a good combined order of variable elimination and conditioning is important for both likelihood computations and haplotyping, since the chosen order has a major effect on both time and memory requirements of the computations. This problem has been addressed quite extensively in the context of genetic linkage analysis. The two main approaches for performing likelihood computations on pedigrees are the Elston-Stewart algorithm [Elston and Stewart, 1971] which peels one nuclear family after another, and the Lander-Green algorithm [Lander and Green, 1987] which peels one locus after another. Another approach, proposed by Lange and Boehnke [1983], is to peel one person after another. These approaches are all variants of variable elimination methods which use different fixed elimination orders. Finding a good elimination order is also essential in a variety of combinatorial problems, such as: constraint satisfaction, independent set, dominating set, graph K-colorability and Hamiltonian circuit [Arnborg, 1985; Dechter, 1998], as well as in other applications of Bayesian networks.

The problem of determining a good combined order of variable elimination and conditioning can be reduced to a graph-theoretic problem, namely, all elimination and conditioning operations are performed on the undirected graph representation of the Bayesian network. The undirected graph representation is obtained from the Bayesian network by connecting each pair of vertices that have edges leading into a common vertex, and removing the directionality of the edges [Pearl, 1988].

When a vertex is eliminated from the graph, its set of neighbors are connected to form a clique. By a clique we mean that every vertex in the set is connected with an edge to every other vertex in the set. The cost of eliminating vertex v from graph G_i is $c_{G_i}(v) = \Pi_{u \in \bar{N}_{G_i}(v)} w(u)$, where $\bar{N}_{G_i}(v)$ represents the set of neighbors of v including v itself, and $w(v)$ is the weight of v, namely, the number of possible values of variable X_v. In the case when there is no memory limitation, we aim to find an elimination order $\hat{X}_\alpha$ which satisfies $\hat{X}_\alpha = \arg\min_\alpha C(X_\alpha)$, where

$$C(X_\alpha) = \sum_{i=1}^{n} c_{G_i}\left(X_{\alpha(i)}\right), \tag{4}$$

and α denotes a permutation on $\{1, ..., n\}$. In eq. (4), G_i, $i = 2, ..., n$ denotes the sequence of residual graphs obtained from a given graph $G_1 = G$ by eliminating its vertices in the order $X_{\alpha(1)}, ..., X_{\alpha(i-1)}$.

This cost function, which is often referred to as the *total state space* (Kjærulff, 1990], is an approximated measure of the time and space complexity of the computations, provided that the heaviest clique created fits into the RAM size of the working environment. If this is not the case, then conditioning is needed and a more elaborate cost function, described in [Fishelson and Geiger, 2003], is required. In this case, we obtain a constrained elimination order $X_{\alpha,\beta} = ((X_{\alpha(1)}, ..., X_{\alpha(n)}), \beta)$, which is a sequence of vertices along with a binary vector β such that vertex $X_{\alpha(i)}$ is eliminated if $\beta_i = 0$ and conditioned on if $\beta_i = 1$.

If we replace the summation in eq. (4) with maximization, namely with $C(X_\alpha) = \max_{1 \le i \le n} c_{G_i}(X_{\alpha(i)})$, then the resulting cost function represents the weight of the heaviest clique created during the elim-

Procedure SG(G, T, C_1, C_2)

Input: A weighted undirected graph $G(V, E, w)$, a threshold T, and two cost functions (C_1, C_2).

Output: An elimination sequence $X_{\alpha,\beta}$ such that the elimination cost of each vertex $\leq$ T.

1. Initialize vector β of size n with zeroes.

2. $i \leftarrow 1$

3. $G \leftarrow G_i$

4. While G_i is not the empty graph **do**

- **forall** $X \in V_i$ compute the cost $C_1(X)$
- Pick 3 vertices $X_{k_1}, X_{k_2}, X_{k_3}$ with a minimum cost C_1.
- Flip a coin, biased according to the costs of the 3 vertices, to choose X_k.
- **If** $C_{G_i}(X_k) > T$ then {**conditioning on** X_k}

  Pick X_k that optimizes C_2 to condition on.

  $\beta_i \leftarrow 1$

  Else {**eliminating** X_k}

  $E_i \leftarrow E_i \bigcup \{(u, v) | u, v \in N_{G_i}(X_k)\}$ (*connect all the neighbors of* X_k)
- Remove X_k and its incident edges from G_i
- $\alpha(i) \leftarrow k$
- $i \leftarrow i + 1$

5. Return $X_{\alpha,\beta} = ((X_{\alpha(1)}, \ldots, X_{\alpha(n)}), \beta)$

Fig. 3. Procedure SG.

ination, which is called the *weighted treewidth* of the graph with respect to the specific elimination order. Our aim is to find an elimination order which produces the lightest heaviest clique among all elimination orders, namely, to find the so called *weighted treewidth* of the graph. If, in addition to replacing the summation with maximization, the weight of all vertices is constant, then this problem is reduced to finding the *treewidth* of the graph, which is NP-complete [Arnborg et al., 1987]. The treewidth of a graph with respect to a given elimination order is the size of the largest clique created during the elimination minus one. The treewidth of a graph is the minimal treewidth over all possible elimination orders for the graph.

We devised a new algorithm for finding a combined order of variable elimination and conditioning and applied it for both haplotyping and likelihood computations. The algorithm is composed of two stages. First, a set of reduction rules are applied on the graph as a preprocessing step. Second, several stochastic-greedy algorithms are applied sequentially to determine an elimination order for the residual graph.

Preprocessing Rules

Eijkhof et al. [2002] present a set of safe reduction rules for the *weighted treewidth* problem. Application of these rules can significantly reduce the size of the graph, without increasing the weighted treewidth of the graph. We tested these rules for our optimization problem and found that two of these reduction rules, the *simplicial* and the *almost simplicial* rules, are worthy to incorporate. The run time of these rules is negligible compared to the total run time for finding an elimination order, and by reducing the size of the graph, each iteration of the stochastic-greedy algorithms applied later is shorter. Throughout the application of the reduction rules, a variable *low* which represents the largest lower bound known for the

weighted treewidth of the original graph is maintained. We denote by $nw(v)$ the product $\Pi_{u \in \bar{N}_{G_i}(v)} w(u)$.

Simplicial Rule. Let v be a *simplicial* vertex in G_i, namely, its set of neighbors form a clique. Recall that a clique is a set of vertices where every vertex is connected to every other vertex. The simplicial rule removes v from the graph, and updates the variable *low*: $low = max(low, nw(v))$.

Almost Simplicial Rule. A vertex v is called an *almost simplicial* vertex in G_i if all its neighbors, except one u, form a clique. Vertex v is removed if $low \geq nw(v)$ and $w(v) \geq w(u)$.

Stochastic-Greedy Algorithms

Three Stochastic-Greedy algorithms for finding a combined order of variable elimination and conditioning, all based on the same common procedure SG (fig. 3), have been incorporated. The input to this procedure is a weighted undirected graph $G(V, E, w)$ resulting from the application of the reduction rules, a threshold T which represents the memory limitation, and two cost functions, C_1 and C_2, that vary between the three algorithms. The threshold T is determined dynamically according to the memory available at run time. According to cost function C_1, the next vertex to eliminate is chosen, and according to C_2, a vertex to condition on is chosen. In each iteration, three vertices with a minimal cost (according to C_1) are selected, and a coin, biased according to the costs of the vertices, is flipped to choose between them. If in iteration i, the weight of the clique created by the elimination of the chosen vertex is above the given threshold, a new vertex is chosen (according to C_2) to condition on rather than eliminate. Procedure SG is run many times, each time finding a new elimination order, and comparing it to the best order found so far. If the cost of the new elimination order is smaller than that of the best previously found order, then the new order and cost are recorded.

The stochastic-greedy algorithms used are: *Min-Weight* (Min-W), *Min-Fill*, and *Weighted Min-Fill* (WMin-Fill). The first two algorithms are based on known cost functions (Kjærulff, 1990], whereas the cost function of the *Weighted Min-Fill* algorithm is new and shows superior performance in many cases, as demonstrated by Experiment E. We now describe the different cost functions. The cost C_1 of eliminating a vertex according to the *Min-Weight* heuristic is the product of weights of its neighbors, whereas the cost of eliminating a vertex according to the Min-Fill heuristic is the number of edges that need to be added to the graph due to its elimination. The Weighted Min-Fill heuristic is a novel modification of the Min-Fill heuristic to a weighted graph. If we define the *weight of an edge* to be the product of weights of its constituent vertices, then the cost of eliminating a vertex according to the Weighted Min-Fill heuristic is the sum of weights of the edges that need to be added due to its elimination. The cost function C_2 for the Min-Fill and Weighted Min-Fill algorithms is the same as the first option described in [Fishelson and Geiger, 2003], i.e.,

$$C_2(X) = \sqrt{\left|N_{G_i}(X)\right|}\, C_1(X),$$

where $N_{G_i}(X)$ represents the set of neighbors of X in G_i, and $|N_{G_i}(X)|$ represents the number of neighbors of X in G_i. The cost function C_2 for the Min-Weight algorithm is the same as the second option described in [Fishelson and Geiger, 2003], i.e.,

$$C_2(X) = \sqrt{\left|f_{G_i}(X)\right|}\, C_1(X),$$

where $f_{G_i}(X)$ are the probability tables that include X (called factors) accompanying graph G_i.

The incorporation of these three algorithms and not others that were tried, such as Maximum Cardinality Search MCS [Tarjan and Yannakakis, 1984] or a weighted version of it (WMCS), is based on the fact that the other algorithms are superior in only a few cases. Neither of the three algorithms that were incorporated is better than the others in all cases, and therefore, each of the algorithms is run a certain percentage of the total optimization time (fig. 4). We denote by %MW, %MF, and %WMF the percentage of iterations spent on running the Min-Weight, Min-Fill, and Weighted Min-Fill algorithms respectively. These percentages have been determined experimentally based on the relative performance of each algorithm. The total number of iterations N is determined according to the complexity of the problem at hand, which is estimated according to the cost of the elimination order found by the *deterministic-greedy* Min-Weight algorithm [Fishelson and Geiger, 2003]. The only difference between the deterministic algorithm and the stochastic algorithm is that in each iteration, the deterministic algorithm chooses to eliminate a vertex with a minimal elimination cost according to the Min-Weight cost function, rather than flip a coin. If the cost of the elimination order found by this deterministic algorithm is lower than some threshold C_{min} then no optimization is performed.

The superior performance of the Min-Fill algorithms, as observed from Experiment E, may stem from the following observation. Assume that there is a set of vertices that almost forms a clique. Despite the fact that the elimination of one of these vertices would add only a few edges to the graph, this vertex would not be a preferred vertex to eliminate by the Min-Weight heuristic if the clique that is created is heavy. Hence, the Min-Weight Heuristic may complicate the given graph by choosing in some early iterations vertices whose elimination creates light cliques but possibly adds many edges to the graph.

Complexity Analysis

As mentioned above, the Elston-Stewart algorithm proceeds by peeling one nuclear family after another, and hence, its complexity is linear in the pedigree size (for sufficiently simple pedigrees), but exponential in the number of loci in the analysis. The Lander-Green algorithm, on the other hand, which proceeds by peeling one locus after another, is exponential in the number of individuals in the pedigree but linear in the number of loci. In the following, we argue that the time complexity of the Elston-Stewart and Lander-Green algorithms is in fact dominated by the sum of sizes of the factors created during the computation, each in a different but predetermined order of variable elimination. Consequently, these algorithms' time complexity is a special case of our algorithm's time complexity, which is also dominated by the sum of sizes of the factors created during the computation. The difference is that our algorithm does not rely on a fixed elimination order, but determines it automatically according to the problem at hand, so as to minimize the sum of sizes of the factors created during the computation, and hence, minimize the run time.

If we were to build a Bayesian network representation of the pedigree according to the Elston-Stewart algorithm, then each person would be represented by a variable in the network. Since each variable represents a multi-locus genotype, the number of possible values of a variable is exponential in the number of loci in the analysis. The elimination order, according to the Elston-Stewart

Algorithm Find-Order(G, T,C_{min})

Input: A weighted undirected graph $G(V, E, w)$, a threshold T, and a minimum cost C_{min}.

Output: An elimination sequence $X_{\alpha,\beta}$ such that the elimination cost of each vertex $\leq$ T.

1. $X_{\alpha,\beta} \leftarrow$ **deterministic-greedy**(G, T)
2. **If** $C(X_{\alpha,\beta}) < C_{min}$ then **return** $X_{\alpha,\beta}$
3. **Set** N (total ♯ of iterations) according to $C(X_{\alpha,\beta})$
   - $I[1] \leftarrow \lfloor \%\mathbf{MW} * N \rfloor$
   - $I[2] \leftarrow \lfloor \%\mathbf{MF} * N \rfloor$
   - $I[3] \leftarrow \lfloor \%\mathbf{WMF} * N \rfloor$
4. **For** $j \leftarrow 1$ to 3 **do**

   For $i \leftarrow 1$ to $I[j]$ **do**
   - Find a candidate order:

     $X^{temp}_{\alpha,\beta} \leftarrow \mathbf{SG}(G, T, C^j_1, C^j_2)$
   - Update the best order:

     If $C(X^{temp}_{\alpha,\beta}) < C(X_{\alpha,\beta})$ then

     $X_{\alpha,\beta} \leftarrow X^{temp}_{\alpha,\beta}$
5. **Return** $X_{\alpha,\beta}$

Fig. 4. Algorithm Find-Order.

algorithm, involves peeling one nuclear family at a time; namely, in each step, we create a factor which includes all variables of a nuclear family. The size of this factor is a product of the number of individuals in the nuclear family and the number of possible multi-locus genotypes for each family member. The complexity of the algorithm is dominated by the sum of sizes of the factors created during the computation, which is on the order of the product of the number of individuals in the pedigree and the number of values of a variable. Hence, the complexity is exponential in the number of loci in the analysis and linear in the number of individuals in the pedigree.

If we were to build a Bayesian network representation of the pedigree according to the Lander-Green algorithm, then each single-locus genotype would be represented by a variable in the network. The elimination order, according to the Lander-Green algorithm, involves peeling one locus at a time; namely, in each step, we create a factor which includes all variables representing a single locus. Hence, the size of the factor is exponential in the number of individuals in the pedigree. The complexity of the algorithm is dominated by the sum of the factors created during the computation, which is on the order of the product of the number of the loci in the analysis (or, equivalently, the number of factors created) and the size of a factor. Namely, the complexity is exponential in the number of individuals in the pedigree and linear in the number of loci in the analysis.

To summarize, the complexity of all three algorithms is dominated by the sum of sizes of the factors created during the computation, which depends on the specific elimination order chosen by each algorithm. If our algorithm chooses an elimination order as in Elston-Stewart, then its complexity is exponential in the number of loci and linear in the number of individuals. If it chooses an elimination order as in Lander-Green, then its complexity is exponential in the number of individuals and linear in the number of loci. For any intermediary order, the complexity is somewhere in the middle.

From a practical point of view, the complexity of our algorithm depends on the size of the pedigree, the complexity of the pedigree, the number of loci in the analysis, the number of typed individuals, and the elimination order found. It is hard to predict in advance the exact complexity of a given problem. In order to assist the user in determining the complexity of a given problem, SUPERLINK prints the complexity class of the elimination order $X_{\alpha,\beta}$ found, which is defined as $\lfloor \log_{10} X_{\alpha,\beta} \rfloor$. If the complexity class is 14 or

higher, then computations are very time consuming or even infeasible. In such a case, the user can use MCMC methods to generate an approximate maximum-likelihood haplotype configuration.

It should be emphasized that with hundreds of SNP markers, it is not feasible to find optimal haplotypes without resorting to the elimination order as in Lander-Green algorithm which works for small pedigrees. For larger pedigrees, approximation algorithms are needed.

Results

We performed several experiments to test the new algorithms. The experiments can be divided into two classes. The first class compares the performance of our haplotyping algorithm to existing haplotyping algorithms. The second class is designed to test the performance of the new optimization algorithm. In all experiments in which pedigree data was simulated, the assumptions underlying the simulation were: Hardy-Weinberg and linkage equilibrium, no mutation and no interference. All input files and results of the experiments are readily available online.

Evaluation of the Haplotyping Algorithm

Experiment A (Simulation Study)

We tested our haplotyping algorithm on a complex pedigree of moderate size (fig. 5). This pedigree was adapted from figure 2 in Lin [1996]. So far, only an approximate haplotype analysis was possible for this pedigree. We simulated a random haplotype configuration for this pedigree using the simulation guidelines described by Lin and Speed [1997], and obtained a maximum likelihood haplotype configuration in several minutes using SUPERLINK. This pedigree consists of 27 individuals and is highly inbred. All individuals, except for those in the first two generations, were typed at 10 polymorphic markers, each with 5 alleles of equal frequencies. The recombination fraction between each pair of consecutive markers was set to 0.05. The progress made in resolving this pedigree can be appreciated by citing Lin and Speed [1997]: 'This is a very complex though moderate sized pedigree, with 10 polymorphic markers, and it does not seem to us to be possible to carry out a haplotype analysis for it with existing non-simulation-based statistical methods.'

We note that GENEHUNTER removes 12 individuals from the pedigree in order to perform the computations. To the best of our knowledge, no previous exact algorithm can produce the maximum likelihood haplotype configuration for this pedigree.

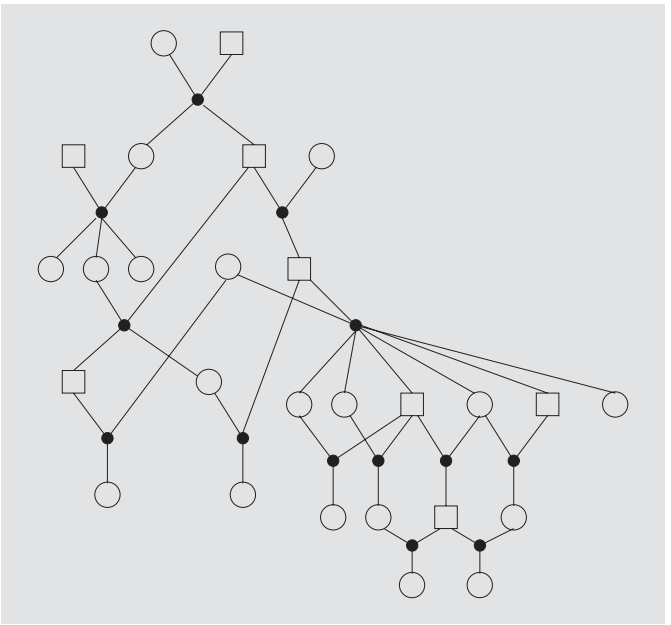

Fig. 5. A complex pedigree of moderate size used for the simulation study in Experiment A. Adapted from figure 2 in Lin [1996].

Experiment B (Testing Correctness)

We tested the correctness of our maximum likelihood haplotyping algorithm by implementing three independent versions of the algorithm, and comparing the results obtained by all three versions. Each version was implemented by different people, to assure an independent evaluation. By correctness we mean that the software finds a haplotype configuration of maximum likelihood given the assumptions of Hardy-Weinberg and Linkage equilibrium. We tested 60 data sets consisting of 5 to 150 individuals and up to 200 markers. In all tested data sets, all three versions produced haplotype configurations with the same likelihood. It should be noted that there is usually more than one maximum-likelihood haplotype configuration, and hence, various algorithms often produce different haplotype configurations.

Experiment C (Testing Accuracy)

Existing approximate methods for haplotyping provide no guarantee on the accuracy of the output. Using our haplotyping algorithm, approximated haplotyping can be compared with the optimal solution on larger pedigrees than was previously possible. This experiment tested the accuracy of a state of the art program that uses MCMC, called SIMWALK2 [Sobel and Lange, 1996]. We tested 75 random data sets consisting of 15 to 50 indi-

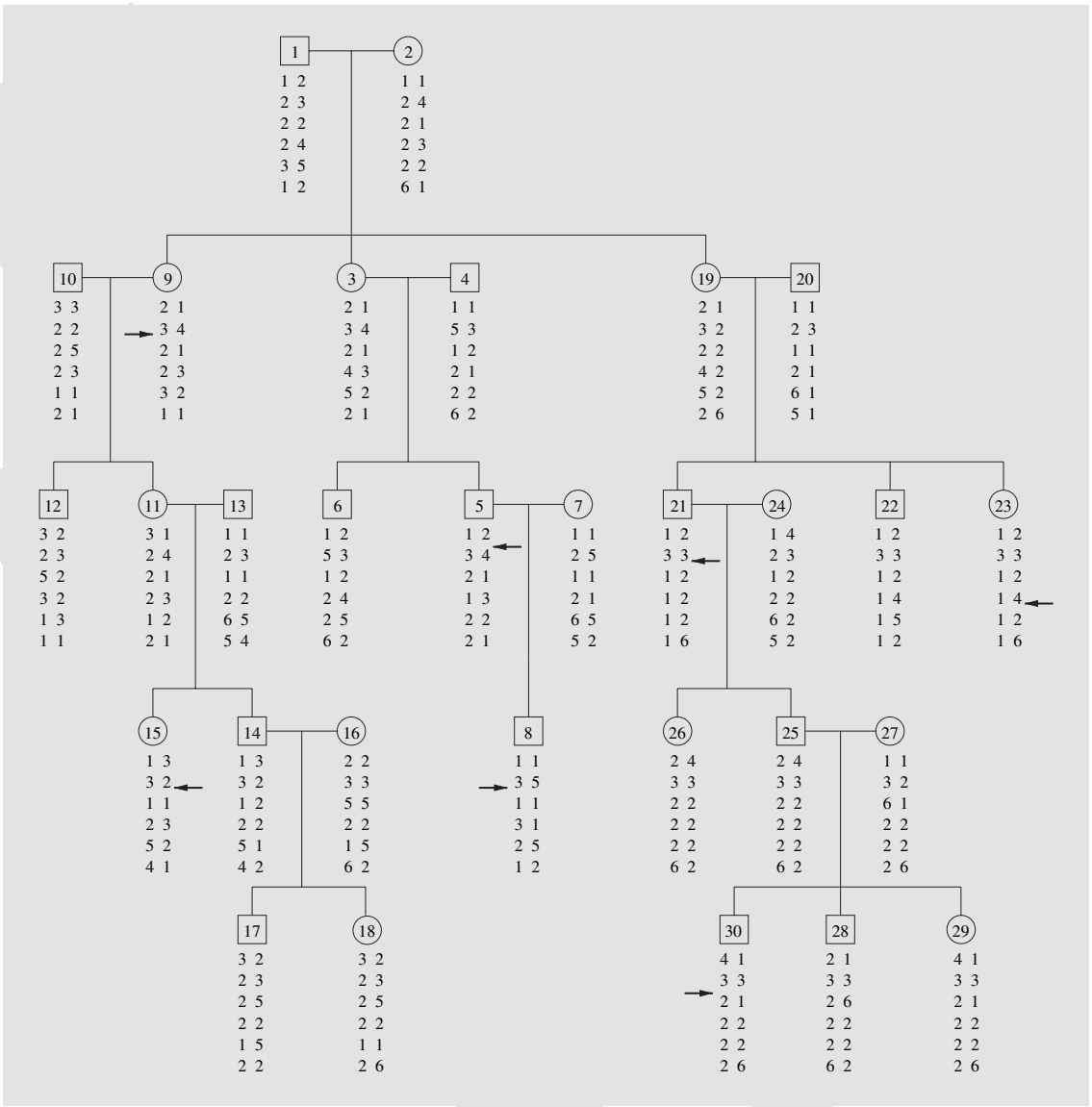

Fig. 6. The maximum-likelihood haplotype configuration obtained by SUPERLINK for one of the datasets in Experiment C (dataset 14). Its $\log_{10}$ likelihood is –121.41774. It is 4.2 times more likely than the haplotype configuration reported by SIMWALK2 (see fig. 7).

viduals and up to 10 markers. SIMWALK2 found a maximum likelihood assignment in 45 out of the 75 data sets. In the other 30 data sets, the average difference in the log-likelihood of the assignment reported by SIMWALK2 compared to the maximum likelihood assignment was merely 1%.

An example for the different outputs of SUPERLINK and SIMWALK2 is shown in figures 6 and 7. These figures show the haplotype configurations obtained by SUPERLINK and SIMWALK2 for one of the data sets used in this experiment (data set 14). This is a loopless pedigree with 30 individuals, typed at 6 polymorphic markers. The recombination fractions between the markers according to their linear order are: 0.154, 0.229, 0.225, 0.194, 0.17. As can be seen from figures 6 and 7, the two haplotype configurations are quite similar. Many of the differences involve different phases in the haplotypes of founders. Such information can not be discerned by the data, and hence, such differences are meaningless. However, the haplotype configuration found by SIMWALK2 contains

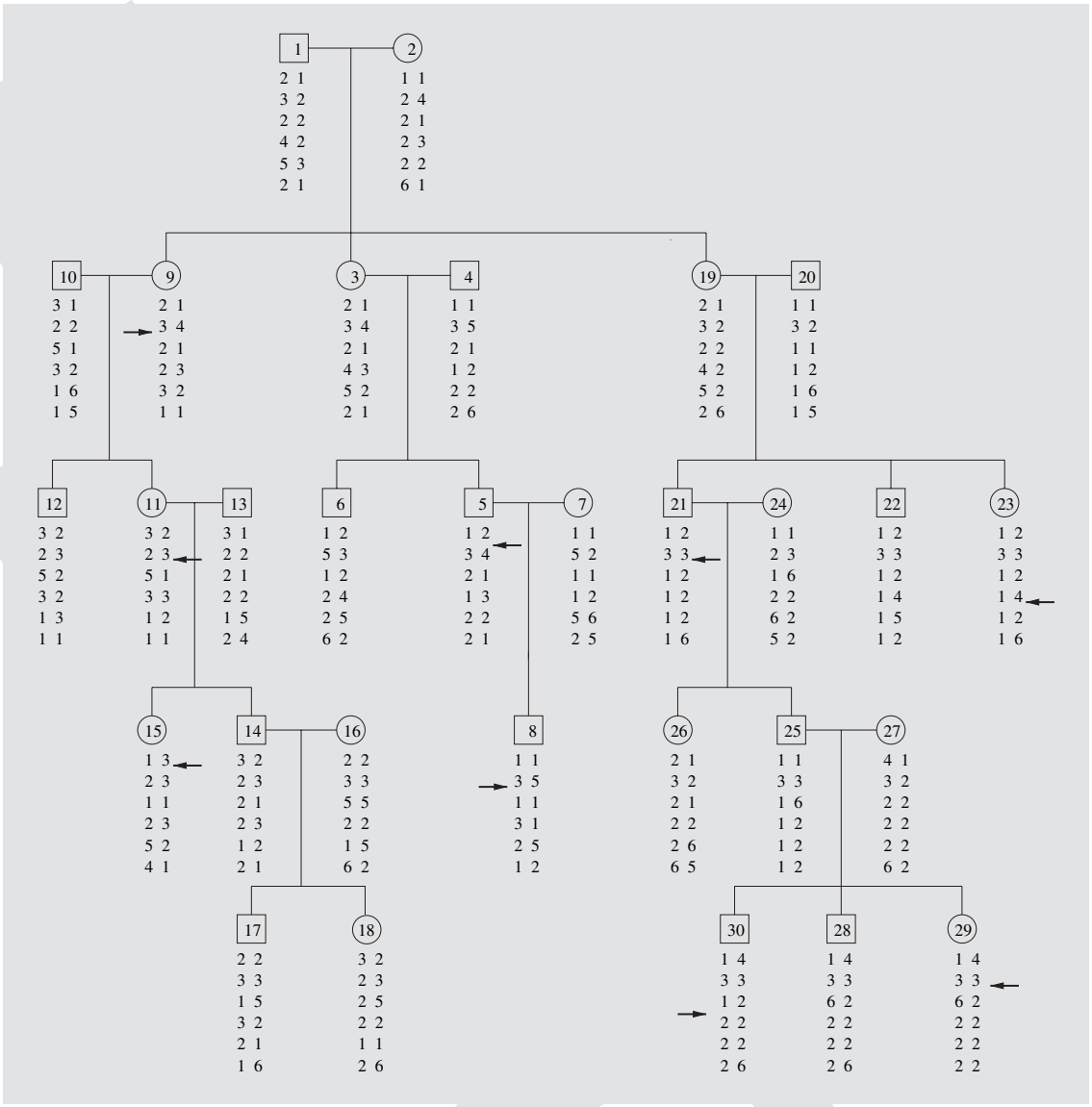

Fig. 7. The haplotype configuration obtained by SIMWALK2 for one of the datasets in Experiment C (dataset 14). The $\log_{10}$ likelihood is –122.04171.

9 recombination events whereas the haplotype configuration found by SUPERLINK contains merely 7 recombination events. The positions of 5 of the recombination events found by both programs are the same. The other 2 recombination events found by SUPERLINK are in different positions than those found by SIMWALK2. The likelihood of the haplotype configuration found by SUPERLINK is 4.2 times higher than the one reported by SIMWALK2. In practice, one can not infer which haplotype configuration is closer to the real configuration; however, the haplotype configuration obtained by SUPERLINK is always a haplotype configuration of maximum-likelihood, whereas SIMWALK2 does not always provide a haplotype configuration of maximum-likelihood, as can be seen from this example.

The MRH software [Qian and Beckmann, 2002] could not run on any of the data sets used in this experiment. We also tested the Block-Extension option of pedphase [Li and Jiang, 2003b], running it 200 times on each data set. Coherent output was obtained in 37.64% of the runs. In those cases where coherent output was produced, SUPERLINK produced haplotype configurations with less

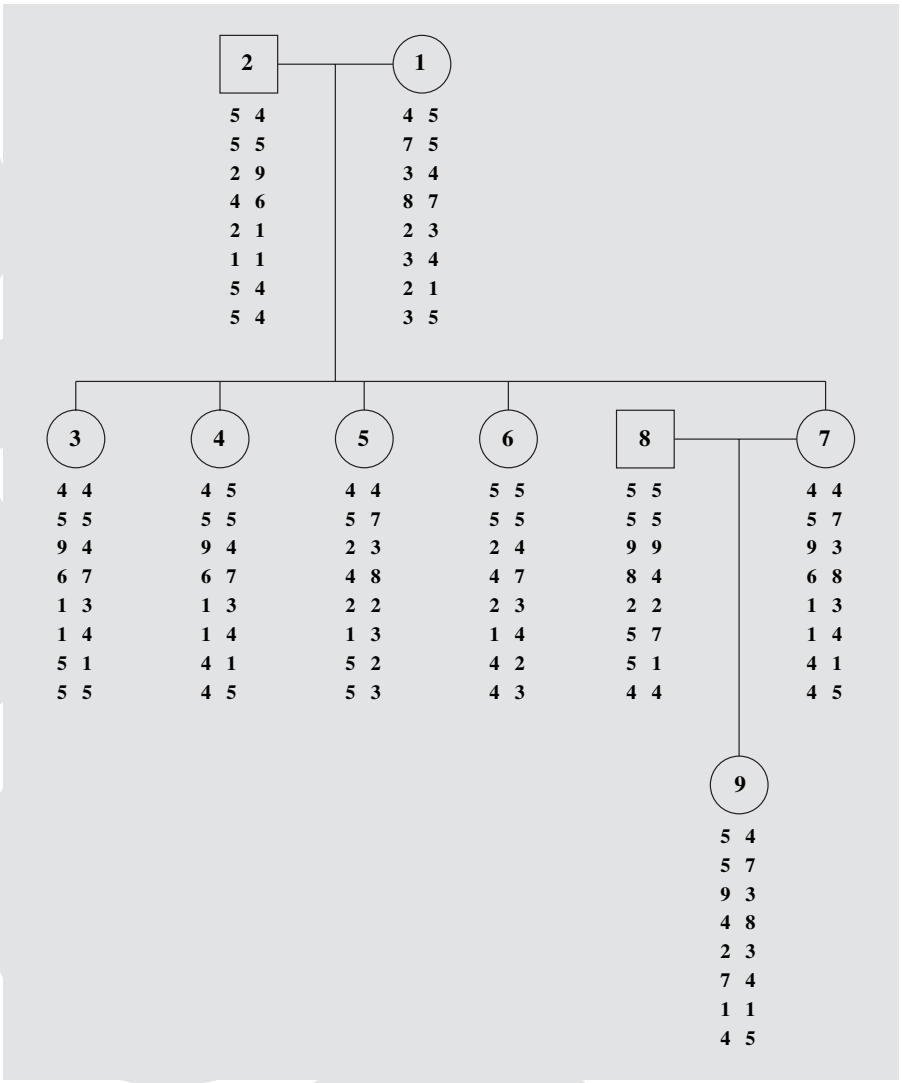

Fig. 8. The maximum-likelihood haplotype configuration obtained by SUPERLINK for the Krabbe dataset [Oehlman et al., 1993].

or equal number of recombination events compared to pedphase. This is interesting to note because the goal of pedphase is to minimize the number of recombination events while superink maximizes the likelihood of data.

Experiment D (Published Disease Data)

We analyzed two published data sets from a study of the Krabbe disease by Oehlman et al. [1993], and from a study on Episodic Ataxia (EA) by Litt et al. [1994]. In both analyses, we assumed linkage equilibrium between the analyzed markers. The Krabbe data set consists of 9 individuals typed at 8 polymorphic markers on chromosome 14. The marker names, according to their linear order are: D14S47, D14S52, D14S43, D14S53, D14S55, D14S48, D14S45, D14S51. The respective sex-averaged recombination fractions between these markers are: 0.1106, 0.1799, 0.0319, 0.0773, 0.0186, 0.1567, 0.0148. The most likely haplotype configuration obtained by SUPERLINK for the Krabbe data set (fig. 8) is identical to the one obtained by MCMC via SIMWALK2 [Sobel et al., 1995; Sobel and Lange, 1996], by Lin and Speed [1997], and by pedphase [Li and Jiang, 2003b].

The Episodic Ataxia data set consists of 29 individuals, which are all typed at 9 polymorphic markers on chromosome 12 except for the first two generation founders. The names of the markers, with respect to their linear order are: D12S91, D12S100, CACNL1A1, D12S372, pY2/1, pY21/1, KCNA5, D12S99, and S12S93. The respective sex-averaged recombination fractions between these markers are: 0.01, 0.01, 0.03, 0.03, 0.01, 0.01, 0.01, and 0.01. The most probable configuration found by SUPERLINK for this data set (fig. 9) is configuration D in

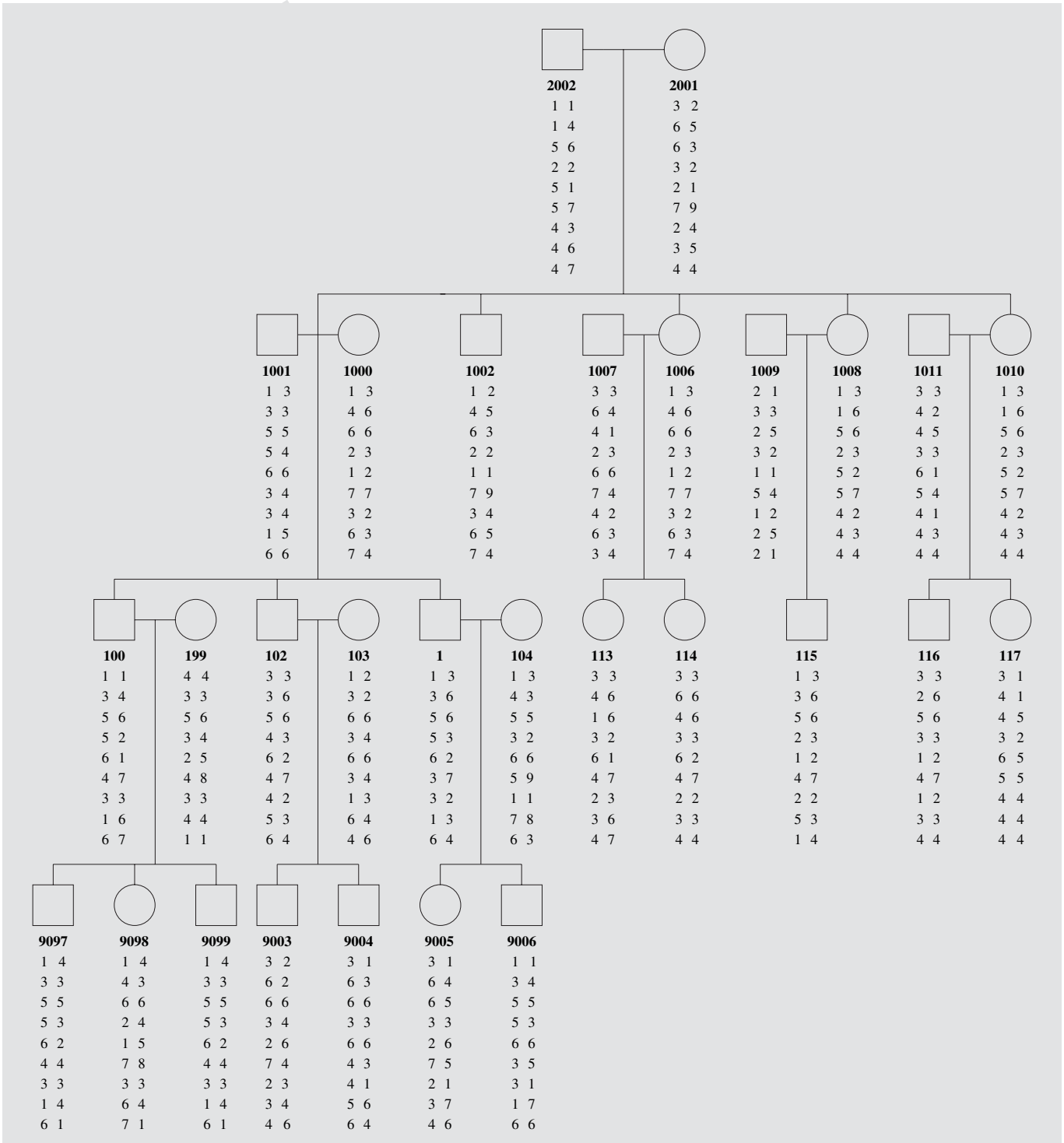

Fig. 9. The maximum-likelihood haplotype configuration obtained by SUPERLINK for the Episodic Ataxia dataset [Litt et al., 1994]. All individuals are typed. Recombination is seen at individuals 100, 113, and 9004.

Table 1. Comparing different stochastic-greedy algorithms

#Loci[a]	#People[b]	%Typed[c]	Min-W[d]	MCS[e]	WMCS[f]	Min-Fill[g]	WMin-Fill[h]
12	25	80	3.97	5.15	5.36	3.94	**3.93**
14	27	63	17.00	15.54	22.27	**14.17**	14.22
16	22	89	5.30	8.08	8.61	**5.22**	**5.22**
20	31	71	10.99	12.62	15.67	10.38	**10.03**
17	29	56	12.77	11.41	20.15	11.14	**10.56**
19	33	95	6.19	7.03	7.55	**5.70**	5.91
18	31	47	8.36	10.27	13.01	8.23	**7.90**
13	30	30	**2.79**	2.81	2.81	2.80	2.80
22	38	86	6.72	9.41	13.85	6.46	**6.37**
20	48	91	9.28	14.73	13.45	7.84	**7.61**

The cost reported is the $\log_{10}$ of the sum of weights of cliques created during elimination. Best results are in bold. WMin-Fill is superior in most cases for a variety of pedigree problems.

[a] Number of loci being analyzed.

[b] Number of people in the pedigree.

[c] Percentage of typed people in the pedigree.

[d] Elimination cost obtained by the Min-Weight algorithm.

[e] Elimination cost obtained by the Maximum Cardinality Search algorithm.

[f] Elimination cost obtained by the Weighted Maximum Cardinality Search algorithm.

[g] Elimination cost obtained by the Min-Fill algorithm.

[h] Elimination cost obtained by the Weighted Min-Fill algorithm.

figure 2 of Qian and Beckmann [2002], which differs from the one obtained by SIMWALK2 [Sobel et al., 1995; Sobel and Lange, 1996] in the position of one recombination event. The only difference is the genotype phase in the fourth marker of individuals 1007 and 113. This configuration is also very similar to the one found by Lin and Speed [1997].

Evaluation of the Optimization Algorithm

Experiment E (Stochastic Algorithms)

This experiment compared the performance of different stochastic-greedy algorithms (table 1). Each of the stochastic algorithms is run for 1000 iterations, after the reduction rules have been applied. We used graphs created from simulated pedigree data. Note that this experiment compares the elimination costs found by the algorithms for the case where $T = \bullet\bullet$, namely no conditioning is performed. The results presented in table 1 are a representative sample from the experiment that was performed on 100 data sets. In 100 data sets, the distribution of algorithms that found the lowest cost was as follows: Min-Weight – 4%, MCS – 9%, WMCS – 7%, Min-Fill – 25%, and Weighted Min-Fill – 76%. Note that these percentages do not sum to 100% since, for some data sets, several algorithms found a minimal cost elimination order. As can be seen, the Min-Fill and Weighted Min-Fill are superior to the other heuristics. However, since the Min-Weight heuristic is the fastest and it works well when conditioning is needed (results not shown), it is profitable to first run it and then run the Min-Fill and Weighted Min-Fill heuristics. The MCS and Weighted-MCS heuristics have been found to hardly contribute when applied after the Min-Weight heuristic and are therefore not incorporated. To summarize, using an algorithm which combines the three algorithms, Min-Weight, Min-Fill, and Weighted Min-Fill, is superior to running only one of them, provided the optimization time is small enough compared to the total run time, as is the case for sufficiently large pedigrees.

Experiment F (Total Run Time)

This experiment compared the run time of likelihood computation with the new optimization algorithm for determining an elimination order presented herein, to the run time with the previous optimization algorithm. The total run time includes both optimization time and inference time. We demonstrate the performance of the optimization algorithm using likelihood computations rather than haplotyping, since haplotyping was not implemented in the previous version of SUPERLINK. We tested 50 randomly simulated data sets chosen so that the run time in the new version is above 10 s and below 10 h. The graph

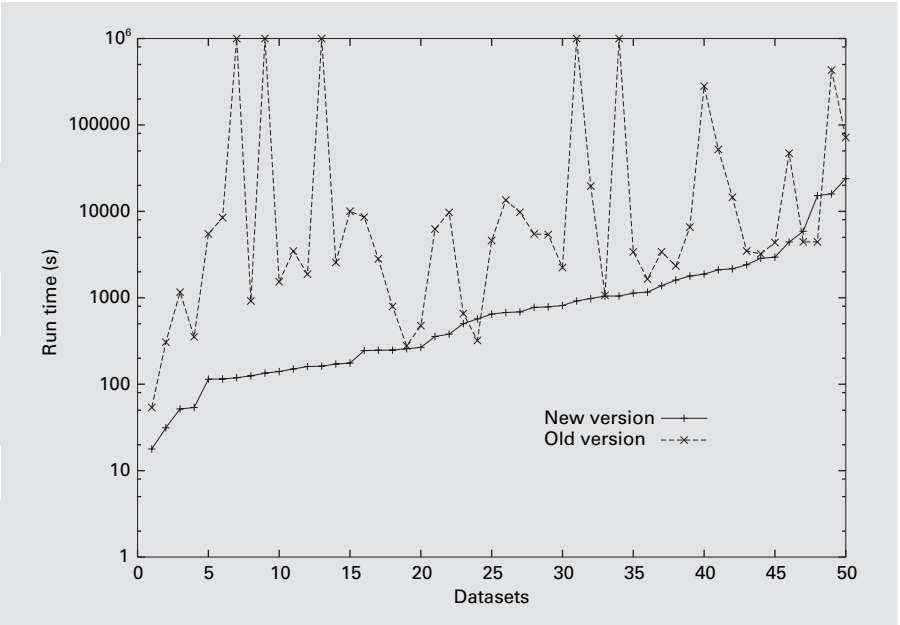

Fig. 10. Run time comparison of likelihood computation using the new optimization algorithm versus the previous optimization algorithm. On each dataset, the old version was run for up to a couple of hours, and the total run time was estimated according to the percentage of the computation that was completed at that time. For those datasets where the estimated run time of the old version was well over 300 h, the run time appears as 10^6 s.

Table 2. Testing some Stochastic-Greedy Algorithms on known benchmarks

Problem	Known cost[a]	Hugin cost[b]	Cost after 100 iterations			Cost after 1000 iterations		
			Min-W[c]	Min-Fill[d]	WMin-Fill[e]	Min-W	Min-Fill	WMin-Fill
Barley	1.71 E07	1.73 E07	1.95 E07	**1.73 E07**	1.82 E07	no change	**1.71 E07**	1.8 E07
Diabetes	9.83 E06	1.04 E07	**1.03 E07**	1.56 E07	1.24 E07	**1.01 E07**	1.38 E07	1.21 E07
Link	2.4 E07	2.62 E07	4.31 E07	**2.76 E07**	3.78 E07	4.15 E07	**2.66 E07**	3.14 E07
Munin1	8.69 E07	1.88 E08	1.84 E08	1.54 E08	**1.39 E08**	no change	**8.76 E07**	no change
Munin2	2.05 E06	2.76 E06	4.05 E06	**3.53 E06**	4.36 E06	3.79 E06	**no change**	4.35 E06
Munin3	3.08 E06	3.24 E06	3.28 E06	3.26 E06	**3.11 E06**	3.27 E06	3.2 E06	**3.11 E06**
Munin4	9.84 E06	1.64 E07	1.72 E07	2.0 E07	**1.37 E07**	1.58 E07	no change	**1.34 E07**
Water	3.03 E06	8.04 E06	3.66 E06	**3.03 E06**	3.47 E06	3.62 E06	no change	**no change**

[a] Best known elimination cost.
[b] Elimination cost found by HUGIN6.1 [Andersen et al., 1989; Hugin, 2002].
[c] Elimination cost obtained by the Min-Weight algorithm.
[d] Elimination cost obtained by the Min-Fill algorithm.
[e] Elimination cost obtained by the Weighted Min-Fill algorithm.

in figure 10 shows that timing was improved in 47 out of 50 data sets, often by an order of magnitude.

Experiment G (Benchmarks)

This experiment (table 2) tested the performance of the three stochastic-greedy algorithms, on eight known benchmarks for Bayesian networks inference. The stochastic algorithms are run after the reduction rules have been applied. We present the best elimination cost found by the algorithms after 100 iterations and after 1000 iterations. Also presented for each benchmark are the best known elimination cost and the elimination cost found by HUGIN6.1 [Andersen et al., 1989; Hugin, 2002], which is a leading software for Bayesian networks. As can be seen, in most cases, running a few iterations of the stochastic-greedy algorithms is superior to the algorithm of HUGIN6.1. Another conclusion that can be drawn from this experiment is that the results of the stochastic-greedy algorithms are comparable to *simulated annealing* [Kjærulff, 1990] which is a very time consuming algorithm, and hence not

Table 3. Reduction rules experiments

#Vertices	#Edges	Using two reduction rules			Using four reduction rules		
		% of vertices eliminated	run time[a]	% of run time[b]	% of vertices eliminated	run time	% of run time
48	126	39.6	0.00	0.00	39.6	0.00	0.00
413	819	19.6	0.00	0.00	19.6	0.01	4.76
724	1,738	53.2	0.00	0.00	53.2	0.02	7.13
189	366	52.2	0.00	0.00	52.2	0.00	0.00
1,003	1,662	68.4	0.00	0.00	68.4	0.03	9.56
1,044	1,745	82.4	0.00	0.00	83.5	0.03	9.18
1,041	1,843	74.0	0.00	0.00	74.0	0.02	6.83
32	123	31.3	0.00	0.00	31.3	0.00	0.00
1,676	2,987	58.0	0.03	5.54	58.0	0.10	18.7
2,449	4,320	57.7	0.06	6.74	57.7	0.34	33.2
3,224	5,641	54.5	0.08	7.14	**54.7**	0.30	16.7
3,254	5,754	59.0	0.06	4.41	59.0	0.30	21.9
4,720	8,211	57.8	0.15	7.58	57.8	0.75	29.8
6,242	10,814	54.7	0.16	4.98	**55.2**	0.63	18.3
4,859	8,517	59.6	0.12	6.85	59.6	0.56	23.6
7,040	12,104	60.0	0.28	9.14	60.0	1.32	31.7
9,276	15,926	55.3	0.30	7.47	**55.6**	1.56	24.3
6,372	11,198	57.8	0.17	5.87	57.8	1.56	24.3
9,292	16,096	57.9	0.37	7.73	57.9	2.23	38.9
12,278	21,215	54.3	0.54	7.78	**54.5**	6.02	73.2

Results in bold indicate problem instances where application of the four reduction rules eliminated more vertices than application of only two rules.

[a] Run time is specified in seconds.

[b] The percentage of time spent on reduction rules out of the time required for finding an elimination order, when merely one iteration of the stochastic-greedy algorithm is performed.

a viable option for minimizing the total run time for genetic linkage analysis. As can be seen from table 2, after 100 iterations the best known cost for the Water problem is matched, and after 1000 iterations the best known cost for the Barley problem is also matched.

Experiment H (Reduction Rules)

This experiment tested the gain due to the reduction rules presented in Eijkhof et al. [2002]. The data sets used are 20 graphs, out of which 12 are created from simulated pedigree data and 8 are known benchmarks for Bayesian networks inference. We have found that two of the reduction rules, the *simplicial* rule and the *almost simplicial* rule, are worthy to apply in almost every problem. Applying these rules usually eliminated between 50 and 60% of the vertices in the data sets (table 3). The other two reduction rules, the *buddies* rule and the *cube* rule, are more time consuming, and applying them does not yield sufficient improvement; therefore we do not use these rules.

Discussion

The use of Bayesian networks enables efficient maximum likelihood haplotyping for more complex pedigrees than was previously possible. We adapted and combined several algorithms, allele recoding, reduction rules, elimination algorithms, and the Elim-Max inference algorithm, into a working system for haplotyping readily available for use by geneticists. This advancement can also be utilized for checking approximate haplotyping algorithms such as MCMC.

Several new features have been incorporated in the algorithm for optimizing the elimination order presented herein. First, reduction rules are applied to speed the computations. Second, several greedy algorithms are run, rather than one. Their allocated relative run time is based on experiments on many Bayesian networks. Third, among the greedy algorithms that are applied, a new greedy algorithm is introduced (Weighted Min-Fill).

Fourth, the threshold which controls the time-space tradeoff is determined at run time according to the memory of the computer, rather than fixed a priori. Our new optimization algorithm is applied whenever the elimination cost found by a quick greedy algorithm is above a certain threshold ($\log_{10} C \geq 9$ in the current implementation). The result is an optimization algorithm which is superior in total run time to the algorithm in Fishelson and Geiger [2003] in 47 of the 50 instances we tried, often by an order of magnitude (Experiment D). The presented order optimization algorithm is used in SUPERLINK for LOD score computations as well as for haplotyping.

Acknowledgments

This research was supported by the Israel Science Foundation. We thank Gil Rubin and Shaul Karni for ideas regarding efficient implementation of our haplotyping algorithm, and Sofia Grinberg and Adi Mano for implementing another version of our algorithm. We thank Uffe Kjærulff and the Research Unit of Decision Support Systems Aalborg University for providing us the datasets and support to conduct experiment G. We also thank Anna Tzemach for computer assistance.

Appendix A

Algorithm Elim-Max (Dechter, 1998)

Input: A Bayesian network $\langle G, P \rangle$; an ordering d of the variables; evidence ε.

Output: The most probable assignment to the variables of the Bayesian network, and its probability.

1. **Initialize:** Generate an ordered partition of the conditional probability tables $\{P_i\}$ into buckets, where bucket B_i contains all the probability tables and evidence whose largest-index variable is X_i.

2. **Backward phase (compilation phase):**
   For $i = n$ to 1, process bucket B_i as follows:
   Let $h_1, \ldots, h_k$ be all the probability tables (new and old) in B_i at the time it is processed, and let $S_1, \ldots, S_j$ be the subset of variables in B_i on which probability tables (new and old) are defined.

   - **If** B_i contains $X_i = a_i$ (X_i is observed), assign $X_i = a_i$ to each h_l, and place it in the bucket of the largest-index variable that appears in its scope.
   - **Else,** $\mathbf{U_i} = \bigcup_{j=1}^{k} S_j - \{X_i\}$. Generate function $h_i = \max_{x_i} \prod_{l=1}^{k} h_l$, and place it in the bucket of the largest-index variable in $\mathbf{U_i}$. Store the optimizing value of X_i for each tuple of $\mathbf{U_i}$, $X_i^{opt}(\mathbf{U_i}) = \arg\max_{x_i} \prod_{l=1}^{k} h_l$.

3. **Store:** the constant computed in B_1. This is the probability of the most probable assignment to the variables of the network.

4. **Forward phase (process X_i after finding an assignment to $X_1, \ldots, X_{i-1}$):**
   For $i = 1$ to n, process bucket B_i as follows:

   - Given the assignment $\mathbf{U_i} = \mathbf{u_i}$, choose $x_i = X_i^{opt}(\mathbf{u_i})$.

5. **Return:** the assignment selected for the variables of the network and its probability.

Appendix B

Proposition: The probability of the haplotype configuration found using allele recoding is the same as the probability of the haplotype configuration found without allele recoding.

Proof: Recall that a haplotype configuration is an assignment of values to all genetic loci variables of individuals in the pedigree. For the proof, we use the word *assignment* rather than haplotype configuration. We divide the set of all possible assignments to two sets: A_1 and A_2. The set A_1 consists of all assignments which fulfill the following conditions: (i) at least one allele assigned to a founder belongs to the set of non-transmitted alleles of this founder, and (ii) this allele is not the one with the highest frequency among the set of non-transmitted alleles of this founder. The set A_2 consists of all other consistent assignments given the data, namely, all assignments which fulfill the following condition: each allele assigned to a founder either belongs to the set of transmitted alleles of this founder, or it is the allele of maximum frequency among the list of non-transmitted alleles of this founder. The proof follows from the following two claims.

Claim 1: The maximum likelihood assignment is in the set A_2.

Proof: Assume to the contrary of the claim that the maximum-likelihood assignment is in the set A_1 and denote it by a_0. Suppose, without loss of generality, that for some founder P in locus i, there are two non-transmitted alleles, N_1 and N_2. Assume that allele N_1 is chosen in assignment a_0, and that $Pr(N_2) > Pr(N_1)$. Since N_1 is non-transmitted, no typed descendant of P inherited this allele from P. Hence, we can replace allele N_1 with allele N_2 in assignment a_0 in all individuals that inherited this allele from P according to a_0, obtaining a consistent assignment denoted by a_1. The only difference in the likelihood of these two assignments is in the frequency of allele N_2 compared to the frequency of allele N_1. Since $Pr(N_2) > Pr(N_1)$, it follows that $Pr(a_1) > Pr(a_0)$ in contrary to the assumption that a_0 is the maximum likelihood assignment.

Claim 2: The probability of each assignment $a \in A_2$ using allele-recoding is the same as without using allele recoding.

Proof: There are three types of probability functions:

1. $Pr(\mathbf{S}, \varepsilon)$ – this function does not change under allele recoding because selector variables do not change.

2. $Pr(\mathbf{F}, \varepsilon)$ – this function is the product of frequencies of all alleles in the genotypes of founders determined by the given assignment. These functions do not change under allele recoding.

3. $Pr(\mathbf{N}|\mathbf{F}, \mathbf{S}, \varepsilon)$ – this probability equals 1 if the assignment is consistent and 0 otherwise. When using allele recoding, it is defined via fuzzy inheritance. If the assignment is consistent without allele recoding, it is consistent also with allelerecoding, since all founders' alleles exist also after recoding by definition of the set A_2. In the case of non-founders, no allele is erased. The alleles are merely divided into groups of transmitted and non-transmitted alleles.

To summarize, all probability functions, whose product constitutes the probability of an assignment $a \in A_2$, remain the same with allele recoding and without allele recoding, and hence, the probability of assignment a is maintained under allele recoding.

Electronic-Database Information

The SUPERLINK program is available as an executable for Linux, Windows and Unix operating systems, at http://bioinfo.cs.technion.ac.il/superlink/, along with user documentation. Also available are the input files and results of all the experiments performed.

References

Abecasis GR, Cherny SS, Cookson WO, Cardon LR: Merlin-rapid analysis of dense genetic maps using sparse gene flow trees. Nat Genet 2002;58:97–101.

Andersen SK, Olesen KG, Jensen FV, Jensen F: HUGIN – a shell for building Bayesian belief universes for expert systems. Proc. of the 11th International Joint Conference on Artificial Intelligence (IJCAI), Vol 2, 1989, pp 1080–1085.

Arnborg S: Efficient algorithms for combinatorial problems on graphs with bounded decomposibility. BIT 1985;25:2–23.

Arnborg S, Corneil DG, Proskurowski A: Complexity of finding embeddings in a k-tree. SIAM J Alg Disc Meth 1987;8:277–284.

Dechter R: Bucket elimination: A unifying framework for probabilistic inference; in J M I (ed): Learning in Graphical Models. Kluwer Academic Press, 1998, pp 75–104.

Eijkhof F, Bodlaender H, Koster A: Safe reduction rules for weighted treewidth. Technical Report 02-49, ZIB, Berlin, Germany, 2002.

Elston RC, Stewart J: A general model for the analysis of pedigree data. Hum Hered 1971;21:523–542.

Ewens WJ, Spielman RS: The transmission/disequilibrium test: History, subdivision, and admixture. Am J Hum Genet 1995;57:455–465.

Fishelson M, Geiger D: Exact genetic linkage computations for general pedigrees. Bioinformatics 2002;18(suppl 1):S189–S198.

Fishelson M, Geiger D: Optimizing exact genetic linkage computations. Proc 7th Conf on Computational Molecular Biology (RECOMB), 2003, pp 114–121.

Friedman N, Geiger D, Lotner N: Likelihood computation with value abstraction. Proc 16th Conf on Uncertainty in Artificial Intelligence (UAI), 2000.

Greenspan G, Geiger D: Model-based inference of haplotype block variation. Proc 7th Conf on Computational Molecular Biology (RECOMB), 2003, pp 131–137.

Gudbjartsson F, Jonasson K, Frigge ML, Kong A: Allegro, a new computer program for multipoint linkage analysis. Nat Genet 2000;25:12–13.

Gusfield D: Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. Proc 6th Conf on Computational Molecular Biology (RECOMB), 2002, pp 166–175.

Hugin ■■: The API reference manual version 5.4, 2002.

Kjærulff U: Triangulation of graph – algorithms giving small total state space. Technical Report R90-09, Department of Computer Science, Aalborg University, Denmark, 1990.

Kruglyak L, Daly MJ, Reeve-Daly MP, Lander ES: Parametric and nonparametric linkage analysis: A unified multipoint approach. Am J Hum Genet 1996;58(6):1347–1363.

Kruglyak L, Lander ES: Faster multipoint linkage analysis using Fourier transform. J Comp Biol 1998;5:1–7.

Lander ES, Green P: Construction of multilocus genetic maps in humans. Proc Natl Acad Sci USA 1987;84:2363–2367.

Lange K, Boehnke M: Extensions to pedigree analysis v. optimal calculation of mendelian likelihoods. Hum Hered 1983;33:291–301.

Lange K, Goradia TM: An algorithm for automatic genotype elimination. Am J Hum Genet 1987;40:250–256.
Lange K, Weeks D, Boehnke M: Programs for pedigree analysis: Mendel, fisher, and dgene. Genet Epidemiol 1988;5:471–473.
Lauritzen SL: Graphical Models. Oxford University Press, 1996.
Li J, Jiang T: Efficient rule-based haplotyping algorithms for pedigree data. Proc 7th Conf on Computational Molecular Biology (RECOMB), 2003, pp 197–206.
Li J, Jiang T: Pedphase: haplotype inference for pedigree data. Submitted, 2003.
Li J, Jiang T: An exact solution for finding minimumrecombinant haplotype configurations on pedigrees with missing data by integer linear programming. Proc 8th Conf on Computational Molecular Biology (RECOMB), 2004.
Lin S: Multipoint linkage analysis via metropolis jumping kernels. Biometrics 1996;52:299–309.
Lin S, Speed TP: An algorithm for haplotype analysis. J Comput Biol 1997;4:535–546.
Litt M, Kramer P, Browne D, Gancher S, Brunt E, Root D, Phromchotikul T, Dubay C, Nutt J: A gene for episodic ataxia/myokymia maps to chromosome. Am J Hum Genet 1994;55:702–709.
O'Connell JR, Weeks DE: The VITESSE algorithm for rapid exact multilocus linkage analysis via genotype set-recoding and fuzzy inheritance. Nat Genet 1995;11:402–408.
Oehlman R, Zlotogora J, Wenger D, Knowlton R: Localization of the krabbe disease gene (galc) on chromsome 14 by multipoint linkage analysis. Am J Hum Genet 1993;53:1250–1255.
Ott J: Analysis of Human Genetic Linkage. Johns Hopkins University Press, Baltimore, 1999.
Park J: Map complexity results and approximation methods. Proc 18th Conf on Uncertainty in Artificial Intelligence (UAI), 2002, pp 388–396.
Pearl J: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Francisco, 1988.
Qian D, Beckmann L: Minimum-recobinant haplotyping in pedigrees. Am J Hum Genet 2002;70:1434–1445.
Schäffer AA: Faster linkage analysis computations for pedigrees with loops or unused alleles. Hum Hered 1996;46:226–235.
Sobel E, Lange K: Descent graphs in pedigree analysis: Applications to haplotyping, location scores, and marker sharing statistics. Am J Hum Genet 1996;58:1323–1337.
Sobel E, Lange K, O'Connell JR, Weeks DE: Haplotyping algorithms. IMA Volumes in Mathematics and Its Applications 1995;81:89–110.
Stephens M, Smith NJ, Donnelly P: A new statistical method for haplotype reconstruction from population data. Am J Hum Genet 2001;68:978–989.
Tarjan RE, Yannakakis M: Simple linear time algorithm to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J Comput 1984;13:566–579.
Thompson EA: Monte Carlo likelihood in genetic mapping. Stat Sci 1994;9:355–366.
Thompson EA, Heath SC: Estimation of conditional multilocus gene identity among relatives; in Seillier-Moseiwitch PDF, Waterman M (eds): Statistics in Molecular Biology. Springer-Verlag, IMS Lecture Note Series, 1999, pp 95–113.
Wisjman P: A deductive method of haplotype analysis in pedigrees. Am J Hum Genet 1987;41:356–373.