Maximum likelihood pedigree reconstruction using integer programming

James Cussens
Dept of Computer Science & York Centre for Complex Systems Analysis
University of York, York, YO10 5DD, UK jc@cs.york.ac.uk

Abstract

Pedigrees are 'family trees' relating groups of individuals which can usefully be seen as Bayesian networks. The problem of finding a maximum likelihood pedigree from genotypic data is encoded as an integer linear programming problem. Two methods of ensuring that pedigrees are acyclic are considered. Results on obtaining maximum likelihood pedigrees relating 20, 46 and 59 individuals are presented. Running times for larger pedigrees depend strongly on the data used but generally compare well with those in the literature. Solving is particularly fast when allele frequency is uniform.

1 Introduction

The problem of finding the most probable pedigree ('family tree') for a group of related individuals, whether human or not, is often needed for paternity and family reunion cases [7]. Correctly specifying relationships is also needed for the proper application of genetic linkage analysis. In the literature the problem is often called *pedigree reconstruction* and we will also make use of this term.

A Bayesian approach is frequently taken where prior knowledge is combined with observed data to define a posterior probability for any given pedigree. Prior knowledge can include information such as known relationships, age and/or sex of some of the individuals and perhaps limits on the number of generations in the pedigree. In addition, probabilistic prior knowledge stating that, for example, very high levels of inbreeding are unlikely, can also be included [7, 13].

Data will be *genotypic* data for each individual under consideration. This data will be defined via a set of *marker loci* each specifying a position on a particular chromosome. The DNA sequence at such loci will vary between different individuals and so the marker can be thought of as a variable. The possible values of this variable are known as *alleles*. Chromosomes come in pairs, one inherited from the father and one from the mother, so there is a pair of allele values, called the *genotype*, for each locus. See [9] for further information. Data for pedigree reconstruction typically consists of genotypes for a number of marker loci: call this a *multi-locus genotype*. In the interests of brevity *multi-locus genotype* will often be abbreviated to just *genotype* in what follows.

As a result of its importance a number of computational techniques have been used for pedigree reconstruction including simple enumeration [7], simulated annealing [2, 11], MCMC [3] and dynamic programming [5]. However, it appears that constraint-based methods have yet to be used for pedigree reconstruction although [12, 10] apply weighted CSP and SAT techniques, respectively, to check the consistency of a given pedigree. Also a weighted MAX-SAT approach has been used for the problem of Bayesian network learning [6]; pedigree reconstruction can be seen as a special case of this (see Section 2).

In this paper pedigree reconstruction is cast as a instance of Bayesian network learning and integer linear programming (IP) is used to search for maximum likelihood Bayesian networks. The paper is structured as follows. In Section 2 a method for representing pedigrees as Bayesian networks (BNs) is given and the likelihood function for such BNs is analysed. Section 3 discusses IP encodings for pedigree reconstruction. Section 4 shows results for the most successful

encoding found to date and the paper ends with conclusions and pointers to future work in Section 5.

2 Pedigrees as Bayesian networks

A Bayesian network (BN) is an acyclic directed graph whose nodes (V) represent random variables. (Such graphs are often called, somewhat imprecisely, directed acyclic graphs (DAGs).) If there is an arrow from node $u \in V$ to node $v \in V$ in the graph then u is said to be a *parent* of v. The parameters of a BN are conditional probability distributions for each node given its parents in the graph. Since the graph is acyclic the product of these conditional probability distributions defines a full joint probability distribution over all random variables represented in the graph.

There are a number of ways of representing pedigrees as BNs [9], but here, like [5], each node in the BN represents a known individual, or more precisely the multi-locus genotype of that individual. An arrow from u to v is a statement that u is the biological parent of v. It follows that no node may have more than two parents. A node with no parents represents a *founder*: an individual neither of whose parents are to be found amongst the individuals considered. A node with one parent represents an individual with exactly one known parent and a node with two parents represents an individual both of whose parents are known. Following [5] Hardy-Weinberg equilibrium will be assumed which implies that the multi-locus genotypes for founders will be probabilistically independent. In addition only complete genotypic data (for a given collection of markers) will be considered.

Following [5] let $\alpha_1(g_v|g_u)$ denote the probability that individual v has genotype g_v given that it has one known parent u with genotype g_u. Let $\alpha_2(g_v|g_u, g_w)$ be the probability that individual v has genotype g_v given that its has two known parents u and w with genotypes g_u and g_w. Let $\alpha_0(g_v)$ be the marginal probability that individual v has genotype g_v. Since for any particular pedigree reconstruction problem the observed genotype g_v for each individual v is fixed, the following notational abbreviation can be introduced.

$$\begin{aligned} \alpha(v, \{\}) &\stackrel{\text{def}}{=} \alpha_0(g_v) \\ \alpha(v, \{u\}) &\stackrel{\text{def}}{=} \alpha_1(g_v|g_u) \\ \alpha(v, \{u, w\}) &\stackrel{\text{def}}{=} \alpha_2(g_v|g_u, g_w) \end{aligned}$$

As noted by [5] due to the assumption of a complete sample, the likelihood of any candidate pedigree G decomposes into a product of conditional probabilities. (The likelihood of G is the probability of the observed data conditional on G being the true pedigree.) Letting $\mathrm{Pa}(v, G)$ denote the parents that $v \in V$ has in a pedigree G, this product can be represented as in (1) and so the log-likelihood, which is more convenient to work with, can be represented as in (2).

$$L(G) = \prod_{v \in V} \alpha(v, \mathrm{Pa}(v, G)) \tag{1}$$

$$l(G) = \log L(G) = \sum_{v \in V} \log \alpha(v, \mathrm{Pa}(v, G)) \tag{2}$$

The problem of maximum likelihood pedigree reconstruction is that of finding G such that $l(G)$ is maximised.

3 An integer programming encoding for ML pedigree reconstruction

In this section a method of encoding the maximum likelihood pedigree reconstruction problem as an *integer programming* problem is presented. A first step towards the encoding is the simple observation that a pedigree specifies the parents, if any, of each individual. So given an individual v, a parent set W and a pedigree G it is determined whether v has parents W in G. This is formalised using the functions $I(W \to v)$ defined in (3), where W is implicitly restricted to be: $W \subseteq V \setminus \{v\}, |W| \leq 2$. This restriction on W will be assumed throughout to simplify the presentation.

$$I(W \to v)(G) = \begin{cases} 1 & \text{if } v \text{ has parents } W \text{ in } G \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The log-likelihood (2) of any pedigree can now be rewritten as in (4).

$$l(G) = \sum_{v,W} \log \alpha(v, W) I(W \to v)(G) \tag{4}$$

Note that in (4), $I(W \to v)(G)$ only takes the value 1 when $W = \mathrm{Pa}(v, G)$. For any other value of W, $I(W \to v)(G) = 0$. The key to the integer programming encoding is to view the $I(W \to v)$ as *binary variables*. Any particular pedigree G determines a joint instantiation of these binary variables, setting exactly $|V| = n$ of these binary variables to 1 and all others to 0. However, most joint instantiations of the $I(W \to v)$ do not correspond to any pedigree. With this in mind the maximum likelihood pedigree reconstruction problem can be reformulated as in (5).

$$\begin{array}{l} \text{Find an instantiation of the } I(W \to v) \text{ which maximises:} \\ \sum_{v,W} \log \alpha(v, W) I(W \to v) \\ \text{subject to the } I(W \to v) \text{ representing a valid pedigree.} \end{array} \tag{5}$$

Because the variables in (5) are integer-valued and the *objective function*

$$\sum_{v,W} \log \alpha(v, W) I(W \to v)$$

is *linear* in these variables this is an *integer linear programming* problem—as long as the necessary constraints on the $I(W \to v)$ can be expressed as linear equations and inequalities. In the following subsections it will be shown that this is indeed the case. ('Integer linear programming' will continue to be abbreviated to 'integer programming' throughout.)

3.1 Constraints

The most basic constraint on the $I(W \to v)$ is that each individual v has exactly one parentset. This can be expressed by n linear equations:

$$\forall v : \sum_{W} I(W \to v) = 1 \tag{6}$$

Any instantiation of the $I(W \to v)$ satisfying the equations given by (6) will represent a graph, but the graph may contain directed cycles. To rule out cycles auxiliary variables are required. There are many ways of ruling out cycles. The following sections present two possible approaches.

3.1.1 Ruling out cycles with a total order

For each distinct pair of individuals u, v a binary variable $I(u < v)$ is created. $I(u < v) = 1$ indicates that u is older than v (u's birth was before that of v). Without loss of generality it can be assumed that no two distinct individuals are of exactly the same age, so that exactly one is older than the other which makes the order on the ages of the individuals a *total order*. This is expressed using the following $n(n-1)/2$ equations:

$$\forall u, v : I(u < v) + I(v < u) = 1 \tag{7}$$

Note that (7) means that half of the $I(u < v)$ variables are effectively redundant. However, it is more convenient to work with the full complement of $I(u < v)$ variables. This does not introduce inefficiency since the IP solver will detect this redundancy and simplify the representation of constraints internally. Note also that in (7) the obvious requirement that $u \neq v$ has not been explicitly stated. This notational convenience will be used throughout: whenever a constraint depends on more than one individual these individuals will be distinct, but this will not be made explicit.

The total order must be transitive (if $u < v$, $v < w$ then $u < w$). This can be represented by the following $n(n-1)(n-2)/3$ constraints:

$$\forall u, v, w : 1 \leq I(u < v) + I(v < w) + I(w < u) \leq 2 \tag{8}$$

Finally, the constraint that parents are older than their children needs to be expressed:

$$\forall u, v : I(u < v) \geq \sum_{W : u \in W} I(W \to v) \tag{9}$$

To see that (9) expresses this relation, suppose that u is a parent of v. In that case exactly one of the $I(W \to v)$ on the RHS of (9) is 1 and thus the RHS is 1. To satisfy the inequality $I(u < v)$ must also be 1.

3.1.2 Ruling out cycles with generation variables

An alternative approach associates a *generation number* with each individual in a pedigree. For a founder this number is zero. For any other individual the generation number is the length of the longest path from a founder to the individual. Let m denote the maximum generation number which is a value set by the user to reflect any prior knowledge. In the absence of such knowledge m takes its maximal value $n - 1$.

Let $\text{gen}(v)$ denote the generation number of individual v. It is not difficult to see that if u is a parent of v then $\text{gen}(v) \geq \text{gen}(u) + 1$. This leads to the following set of $n(n-1)$ constraints:

$$\forall u, v : \text{gen}(v) - \text{gen}(u) \geq -m + (m+1) \sum_{W : u \in W} I(W \to v) \tag{10}$$

To understand (10) observe that if u is not a parent of v then the sum on the RHS is zero and the constraint becomes vacuous. If u is a parent of v then the sum is 1 and so the entire RHS becomes 1, effecting the desired constraint. To see that (10) suffices to rule out cycles note that if w is an ancestor of v then $\text{gen}(w) < \text{gen}(v)$. If a cycle obtains, at least two individuals are their own ancestors and the obvious inconsistency arises. Thus as long as (10) is respected no cycles are possible.

Note that (10) does not fix the values of the $\text{gen}(v)$ variables to their correct values. To see this, suppose that n were, say, 10 and m set to 9, reflecting an absence of domain knowledge.

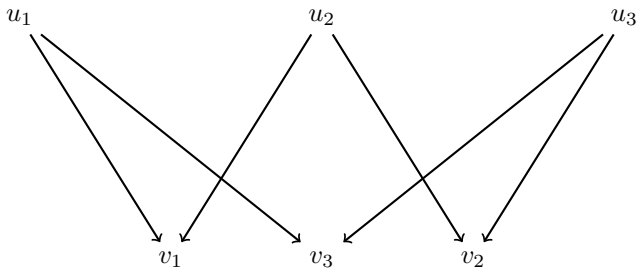

Figure 1: A sex-inconsistent pedigree. It is not possible to consistently assign a sex to each individual.

Suppose that an optimal pedigree were found with the highest valued generation variable having value 5. It would be possible to increase each generation variable by one without violating (10). If our only concern is to rule out cycles this is not a problem, but if it is necessary to ensure that the gen(v) variables take on their correct values then additional constraints placing upper bounds on gen(v) are required.

3.1.3 Ensuring sex-consistency

Any instantiation of the $I(W \to v)$ variables satisfying constraints (6–9) or alternatively (10) will specify an *acyclic* directed graph where each vertex has at most two parents, but not all such graphs represent pedigrees. It is also necessary to ensure that a sex can be assigned to each individual in a consistent manner. An example of an acyclic directed graph where this is *not* the case can be seen in Fig 1. Note that this example was also given by [5]. Call such pedigrees *sex-inconsistent*.

To rule out sex-inconsistent pedigrees another n auxiliary binary variables $I_f(v)$ are created. $I_f(v) = 1$ states that individual v is a female. Constraint (11) states that if an individual v has two parents at most one is female and constraint (12) states that at least one is female. Note that in both cases, if $I(\{u, w\} \to v) = 0$ then the constraints are vacuously satisfied.

$$\forall u, v, w : I(\{u, w\} \to v) + I_f(u) + I_f(w) \leq 2 \tag{11}$$

$$\forall u, v, w : I(\{u, w\} \to v) - I_f(u) - I_f(w) \leq 0 \tag{12}$$

With all these constraints in place, the maximum likelihood pedigree reconstruction problem can be restated as follows:

$$\begin{array}{l} \text{Maximise: } \sum_{v,W} \log \alpha(v, W) I(W \to v) \\ \text{subject either to (6–9,11,12) or (6,10,11,12).} \end{array} \tag{13}$$

4 Results

All results shown here were produced using a 3GHz dual-core Linux machine with the Gurobi IP solver [8]. A number of tests (not reported here) have also been done with SCIP [1] which produced respectable running times which were nonetheless clearly longer than those produced by Gurobi (which automatically parallelises solving on multi-core machines).

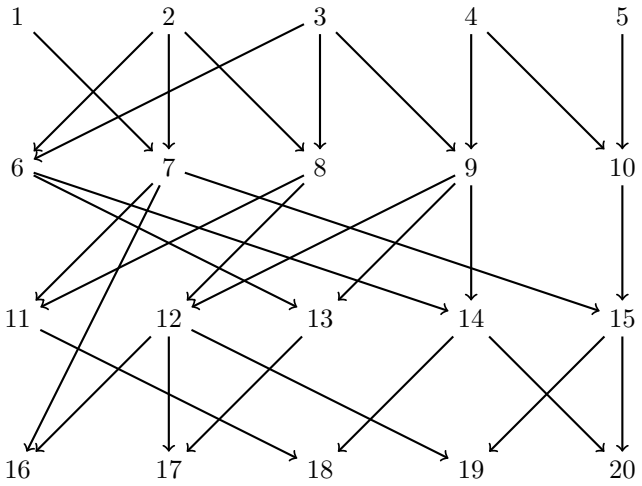

Figure 2: Pedigree of 20 individuals. This pedigree is [5, Fig 3].

Only synthetic genotype data sampled from test pedigrees has been used. This sampling process mimics the inheritance of genotypes from parent to offspring, which is probabilistic, and thus different datasets will be sampled from a given pedigree depending on which random seed is being used. Data was created in this way using the the C++ program `pedsim` used to produce the results in [5]. `pedsim` was also used to compute the log conditional probabilities $\log \alpha(v, W)$ from each of these synthetic datasets, and then to remove $\log \alpha(v, W)$ scores where there exists a higher $\log \alpha(v, W')$ score with $W' \subset W$. Such scores and their accompanying $I(W \to v)$ variables are not needed since in such a case v would never have parents W in the maximum likelihood pedigree. A Python script (available on request from the author) was used to read in the (filtered) $\log \alpha(v, W)$ scores from `pedsim`. Gurobi's Python interface was then used to define and solve the optimisation problem (13). The following sections present results for a number of synthetic pedigree reconstruction problems.

4.1 Pedigree reconstruction for 20 individuals using a total order

The data for this experiment was chosen to be the same (modulo sampling variation) as that of one of Cowell's [5]. The same software (`pedsim`) was used to generate genotypic data from one of Cowell's test pedigrees (Fig 2). Ten marker loci were used. Although the data is synthetic, the markers correspond to real ones. Allele values and founder allele frequencies were taken from [4].

A thousand datasets were sampled and the time taken to find a maximum likelihood sex-consistent pedigree in each case was recorded. Total order constraints were used in each case. The mean solving time was 0.65 seconds with the slowest run taking 17.3 seconds. Only 8 runs took longer than 4 seconds.

4.2 Pedigree reconstruction for 46 individuals using a total order

An experiment identical to that described in Section 4.1 except using a test pedigree of 46 individuals was then carried out. This pedigree was created by editing the source of Cowell's `pedsim` program and is displayed in Fig 3. Only 10 runs were attempted since solving time is

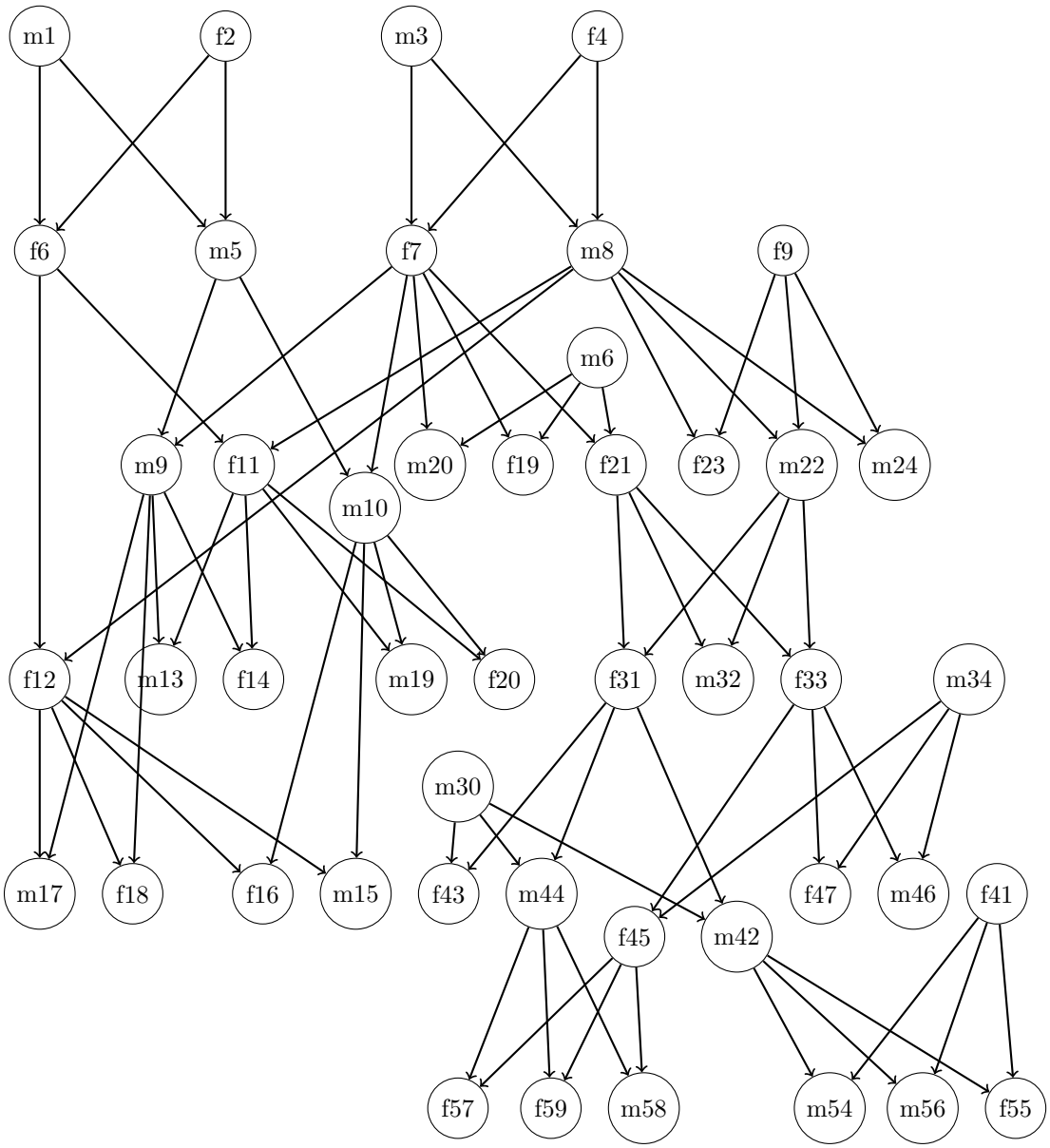

Figure 3: Test pedigree of 46 individuals.

substantially higher than for the 20 individuals case. The solving times for the first 9 runs were, in decreasing order: 20,566s, 7,470s, 4,266s, 1,175s, 722s, 669s, 115s, 59s and 51s. The 10th run was abandoned after failing to identify the maximum likelihood pedigree after 44,135s. The very high variation in solving times is notable. Most datasets produced optimisation problems which could be solved in a reasonable time, but in some cases solving was unacceptably lengthy. Problems of this size are too large for the approach of [5].

Two further experiments were conducted for the 46-individual pedigree. In the first an extra

constraint specifying that there must be at least 20 founders in the pedigree was added. 80 runs were done with this extra constraint. The mean solving time was 18 seconds with 75% of runs below 20 seconds and the slowest taking 128 seconds. In the second experiment a more reasonable constraint on founders was used: that the number of founders was between 10 and 20. 100 runs were done with this constraint. The mean solving time was 34s, with 75% within 30s and the slowest taking 613 seconds.

4.3 Pedigree reconstruction for 46 individuals using generation variables

Although using a total order to rule out cycles in pedigrees produced acceptable results for small numbers of individuals, it is clear that for bigger problems finding a maximum likelihood pedigree is unacceptably slow. Fortunately, switching to using generation variables to rule out cycles as described in Section 3.1.2 results in a significant speed up.

In an initial experiment 100 synthetic datasets were generated from the 46-individual pedigree shown in Fig 3. 19 runs completed successfully, taking a mean time of 432 seconds, but a median time of only 2.2 seconds. The distribution of solving times for these 19 runs was thus highly skewed with the five longest runs taking 7349, 810, 20, 9 and 4 seconds and each of the 6 quickest taking less than a second. However, on the 20th run, Gurobi ran out of memory.

This problem could probably be addressed by instructing Gurobi to use the hard disk when (RAM) memory is exhausted, but this would lead to much slower solving. Instead an extra constraint was added in the hope of both speeding up solving and reducing memory consumption. This constraint stated that in each pedigree there is at least one founder. Since this is always true (due to the acyclicity of pedigrees) a maximum likelihood pedigree will still be returned, but hopefully more quickly. This *founder constraint* is formally expressed as follows:

$$\sum_{v} I(\{\} \rightarrow v) \geq 1 \tag{14}$$

With the founder constraint added 100 runs were attempted (i.e. 100 datasets were simulated and maximum likelihood pedigrees were found for each) and all completed successfully. The mean solving time was 195 seconds and the median was 3.8 seconds. As usual there was therefore a highly skewed distribution of solving times with the ten slowest runs taking the following number of seconds: 8181, 7361, 1638, 830, 309, 249, 127, 118, 110 and 45.

To investigate the effect of increasing the lower bound on the number of founders above one, a particular dataset simulated from the pedigree in Fig 3 was used. This dataset was chosen since it is one of the 'harder' ones resulting in reasonably long solving times. A maximum likelihood pedigree for this data is shown in Fig 4.

As Table 1 makes clear, increasing the lower bound on the number of founders makes a big difference in the time it takes to find a maximum likelihood pedigree. Note, from Fig 4, that at least one maximum likelihood pedigree has 8 founders (m1, f2, m3, f4, f9, m6, f41 and m30). Using 8 as a lower bound on the number of founders solving takes only 3 seconds. Higher lower bounds reduce the solving time further but the pedigree returned is no longer of maximum likelihood as shown by the third column in Table 1. Importantly, raising the lower bound from 0 (which amounts to removing the constraint) to 1 reduces the solving time drastically. Also, interestingly, using lower bounds of 2, 3 or 4 actually increases the solving time compared to a lower bound of 1, but all are still quicker than using no lower bound.

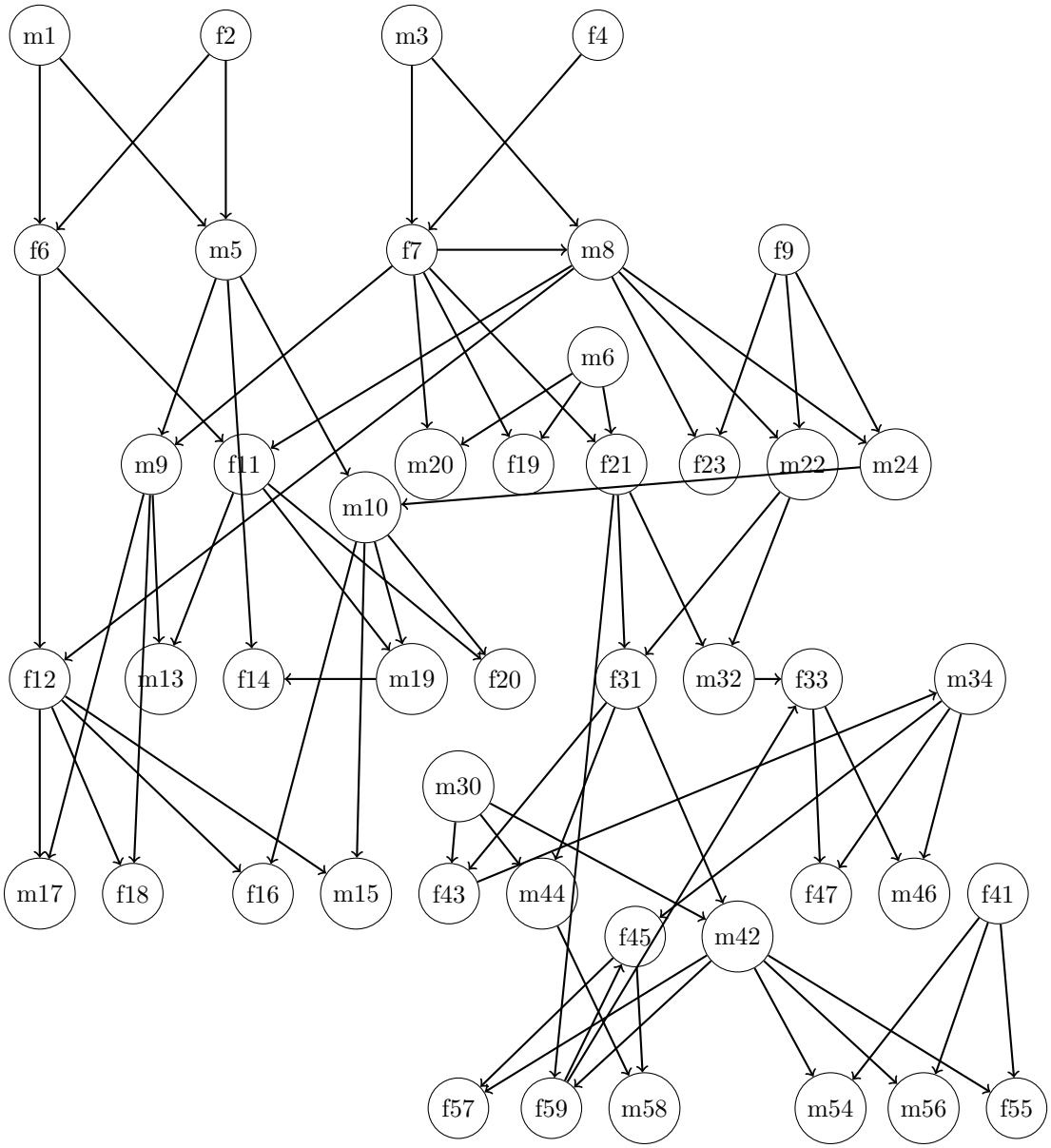

Figure 4: A maximum likelihood pedigree of 46 individuals for a particular dataset simulated from the pedigree in Fig 3

LB	Time in seconds	Likelihood
0	3189.75858617	-6.3680054000e+02
1	1050.95497108	-6.3680054000e+02
2	1695.93844795	-6.3680054000e+02
3	2350.830338	-6.3680054000e+02
4	1220.98797202	-6.3680054000e+02
5	431.202931166	-6.3680054000e+02
6	246.708929062	-6.3680054000e+02
7	63.2229361534	-6.3680054000e+02
8	3.00327396393	-6.3680054000e+02
9	0.523219823837	-6.3933824000e+02
10	0.293282032013	-6.5144025000e+02

Table 1: Solving times for different lower bounds on the number of founders. 'Likelihood' is the likelihood of a maximum likelihood pedigree with the given bound.

4.4 Pedigree reconstruction for 59 individuals using generation variables

An experiment was done to provide as direct a comparison as possible with Almudevar's simulated annealing approach [2]. Datasets were simulated from Almudevar's 59 individual pedigree [2, Fig 2] and as in that paper ten marker loci were used each with 8 equally frequent alleles. Generation variables were used to rule out cycles and the (always admissible) constraint that there is at least one founder was used.

Maximum likelihood pedigrees were obtained from 1000 simulated datasets. The mean solving time was 0.44846 seconds, the median 0.2350 and 75% of runs completed within 0.43860 seconds. A few much longer runs occurred, with one of length 15.26 seconds. The distribution of the 1000 solving times is shown as a box plot in Fig 5 It is notable that solving times are substantially faster on this 59 individual pedigree than for the 46 individual pedigree previously discussed. This is most probably due to the assumption of *equally frequent* (i.e. equally probable) alleles for each of the ten marker loci. This means that genotypic data is more informative than is the case where the distribution is (realistically) skewed as is the case with Cowell's ten marker loci data (Markus Riester, personal communication). Comparing running times to Almudevar [2], there it is stated that for the quickest configuration the time taken for the simulated annealing algorithm to converge "was approximately 6.6 min using a standard personal computer". Note also that simulated annealing does not guarantee that the pedigree found has maximal likelihood.

5 Conclusions and future work

Results on finding pedigrees which are guaranteed to have maximal likelihood using IP have been presented in this paper. The results compare favourably with others in the literature as regards scalability, speed and ensuring sex-consistency (however Riester *et al* [11] report on reconstructing pedigrees from thousands of individuals using simulated annealing).

In this paper the focus is on how best to do maximum likelihood pedigree reconstruction, but there is also the entirely distinct question of whether maximising likelihood is the best way to reconstruct pedigrees. With large amounts of data maximum likelihood usually provides

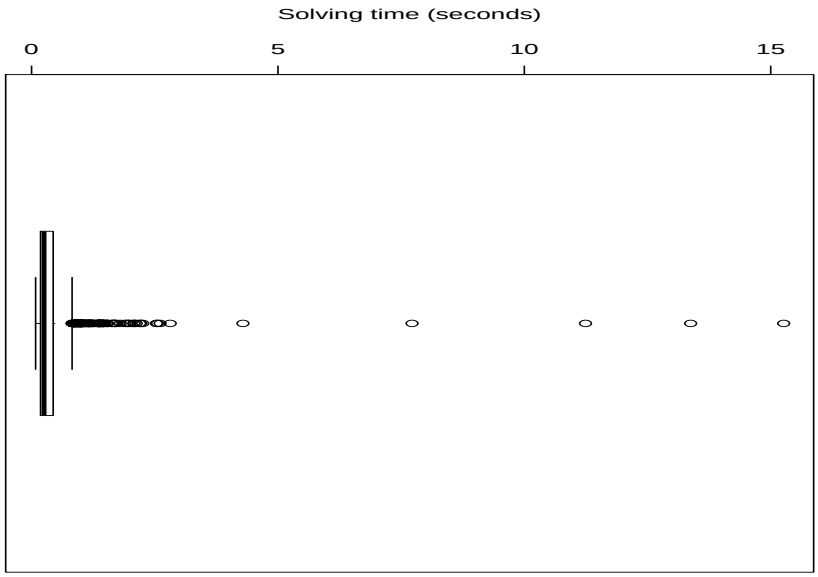

Figure 5: Boxplot representation of the distribution of 1000 maximum likelihood pedigree reconstruction solving times for datasets simulated from Almudevar's 59 individual pedigree.

a reasonable estimate of the true pedigree. So, for example, comparing the 1000 maximum likelihood pedigrees discussed in Section 4.4 to the true data-generating pedigree [2, Fig 2] we find that 533 of them are exactly equal to the true pedigree. The full distribution of parent assignment errors is shown in Fig 6.

Nonetheless the alternative Bayesian approach allows the incorporation of domain knowledge and allows a principled way of quantifying the uncertainty inherent in pedigree reconstruction (*model uncertainty*). In an IP formulation the prior distribution is represented by incorporating extra terms in the objective function. Model uncertainty is addressed by finding many distinct high probability pedigrees rather than returning a single one. Work is currently underway on such a Bayesian approach.

Acknowledgements

Thanks to Robert Cowell, Nuala Sheehan and Markus Riester for useful expertise on pedigrees. Many thanks to Robert Cowell for supplying the `pedsim` software. Thanks also to the anonymous reviewers for their comments and suggestions.

References

[1] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, TU Berlin, July 2007.

[2] Anthony Almudevar. A simulated annealing algorithm for maximum likelihood pedigree reconstruction. *Theoretical Population Biology*, 63:63–75, 2003.

[3] Anthony Almudevar. A graphical approach to relatedness inference. *Theoretical Population Biology*, 71:213–229, 2007.

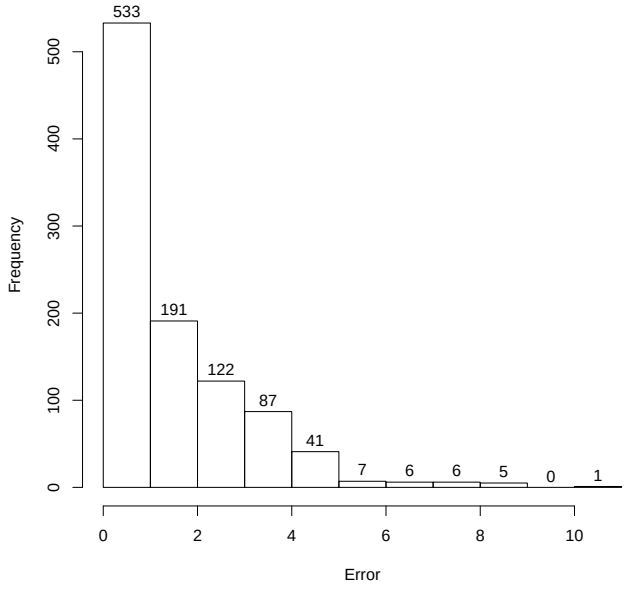

Figure 6: Distribution of parent assignment errors for 1000 pedigrees constructed from data generated from the 'true' pedigree [2, Fig 2]

[4] J.M. Butler, R. Schoske, P.M. Vallone, J.W. Redman, and M.C. Kline. Allele frequencies for 15 autosomal STR loci on U.S. Caucasian, African American and Hispanic populations. *Journal of Forensic Sciences*, 48(4), 2003.

[5] Robert G. Cowell. Efficient maximum likelihood pedigree reconstruction. *Theoretical Popluation Biology*, 76(4):285–291, December 2009.

[6] James Cussens. Bayesian network learning by compiling to weighted MAX-SAT. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI 2008)*, pages 105–112, Helsinki, 2008. AUAI Press.

[7] T. Egeland, P. F. Mostad, B. Mevåg, and M. Stenersen. Beyond traditional paternity and identification cases: Selecting the most probable pedigree. *Forensic Science International*, 110:47–59, 2000.

[8] Gurobi Optimization Inc. *Gurobi Optimizer Reference Manual*, 2010. Version 3.0.

[9] Steffen L. Lauritzen and Nuala A. Sheehan. Graphical models for genetic analyses. *Statistical Science*, 18(4):489–514, 2003.

[10] Panagiotis Manolios, Marc Galceran Oms, and Sergi Oliva Valls. Checking pedigree consistency with PCS. In *Thirteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007)*, number 4424 in LNCS, pages 339–342. Springer, 2007.

[11] Markus Riester, Peter F. Stadler, and Konstantin Klemm. FRANz: reconstruction of wild multi-generation pedigrees. *Bioinformatics*, 25(16):2134–2139, 2009.

[12] Marti Sanchez, Simon de Givry, and Thomas Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13:130–154, 2008.

[13] N A Sheehan and T Egeland. Structured incorporation of prior information in relationship indentification problems. *Annals of Human Genetics*, 71:501–518, 2007.