Selecting Loop Breakers in General Pedigrees

(Technical Report MIP03-24, Labo MIP, Université Paul Sabatier, 31062 Toulouse Cedex 4, France. August 2003. To appear in Human Heredity.)
Short title: Selecting Loop Breakers in Pedigrees

Zulma G. Vitezica[a], Marcel Mongeau [b], Eduardo Manfredi[a], Jean-Michel Elsen[a]

[a] Station d'Amélioration Génétique des Animaux, Institut National de la Recherche Agronomique, BP 27, 31326 Castanet-Tolosan Cedex, France
Telephone: (33) 5 61 28 51 85 - Fax (33) 5 61 28 53 53.
E-mail: *vitezica@toulouse.inra.fr*

[b] Labo Mathématiques pour l'Industrie et la Physique, UMR CNRS 5640, Université Paul Sabatier, 31062 Toulouse Cedex 4, France.

Abstract

The presence of loops in pedigrees poses severe computational problems in likelihood calculation that can be solved by creating an equivalent unlooped pedigree. We introduce a heuristic polynomial-time dynamic-programming algorithm, called SFH, that addresses the problem of selecting a minimal-cost set of loop breakers. We report computational experiments on simulated pedigrees with up to 1000 individuals and 361 loops, and multiple marriages. We compare the loop-breaker set selected by our method with that obtained using the software package FASTLINK 4.1P. Our approach outperforms FASTLINK 4.1P on the computational-time point of view, on the point of view of quality of the loop-breaker set obtained, and on the point of view of the size of the problem that can be addressed.

Key words

Loops breakers, graph theory, cycles, dynamic programming, pedigree.

Introduction

The presence of loops complicates the computation of probability functions in complex pedigrees. The traditional approach to calculate the likelihood involves an algorithm, known as *peeling* [1], whose application is straightforward in pedigrees without loops. To overcome the problem of loops, one strategy consists in cutting explicitly the loops in the

pedigree [2]. A second strategy involves an extension of the basic Elston-Stewart [1] approach to complex pedigrees, which does not require loop cutting [3, 4]. A third strategy consists in cutting the loops, and extending the pedigree at the cuts [5].

The extension of the peeling algorithm proposed by Lange and Elston [2] removes a loop by "cloning" one of the participating individuals. This virtual cloning consists in fact in creating, in pedigree, one copy of an individual. The cloned individual is called a *loop breaker*. This approach requires:

1) identifying individuals participating in loops;

2) selecting a set of the loop breakers;

3) cloning each of them (copying the genetic and phenotypic information);

in order to get an unlooped (post-cloning) pedigree.

Methods for finding and cutting loops have been developed for human pedigrees [6, 7, 8], and for complex animal pedigrees [5, 9]. The method introduced by Xie and Ott [6] uses a graph strategy, called depth-first search, that examines exhaustively the whole pedigree in order to identify a set of loops. The procedure applied to animal pedigrees [9] finds and cuts the loops by considering the pedigree as a sequence of nuclear families. Then, the method traces the succession of families using a recursive procedure. If a family is marked twice, a loop is identified, and an artificial clone is introduced to cut the loop. This method eliminates all the loops after a finite number of steps, and results in the creation of an unlooped pedigree that replaces the original one. The main problem with this approach is that the loops are cut in an arbitrary manner. Arbitrary cutting may be inefficient in situations where, for instance, two clones are created to cut two loops, when creating a single clone would cut both of them. Secondly, arbitrary cutting does not take into account the fact that cloning an individual with known genotype is cheaper in terms of the resulting likelihood computation scheme.

Recently, Becker *et al.* [7] treated the problem of selecting loop breakers. They showed, as did also O'Connell and Weeks [10], that the selection of loop breakers has a significant impact on the computational running time of the subsequent likelihood computation. Accordingly, they proposed an algorithm that attempts at minimizing *both* the number of loop breakers *and* the total number of their associated possible genotypes. The algorithm is optimal for pedigrees with simple marriages (i.e. each individual is involved in at most one marriage); it is an approximation, when multiples marriages are involved. The

software package FASTLINK 4.0P [11, 12] includes the algorithm of Becker *et al.* [7]. Other work by Becker *et al.* [8] presents a heuristic method based on a randomized optimization algorithm to find the loop breakers in pedigrees with multiples marriages. This latter method has been introduced in FASTLINK 4.1P.

We introduce in the present paper an algorithm for selecting loop breakers in general pedigrees. Our algorithm takes into account the combination (chosen by the user) of *both* criteria: *number* of loop breakers *and* total number of associated possible genotypes. Our algorithm is optimal for single marriages and it is an approximation otherwise. However, our approach compares very favourably with the method implemented in FASTLINK 4.1P. First, we recursively remove the leaves of the pedigree graph to obtain a sub-pedigree graph (in an analogous manner as how Becker *et al.* in [7] pre-process the pedigree graph). Secondly, we build an auxiliary graph from this sub-pedigree graph, by cloning the individual nodes, and adding a star (in graph-theory terminology) linking them. Thirdly, we introduce a dynamic-programming algorithm, performed on the auxiliary graph, in order to build the set of loop breakers for the original pedigree. To conclude, we present a computational comparison of our approach with the method implemented in FASTLINK 4.1P on simulated pedigrees with up to 1000 individuals, 361 loops, and multiple marriages.

1. Pedigree graphs

A possible representation of a pedigree is a graph that involves nodes connected by edges arranged in different ways. For example, Figure 1 displays the classical representation of a looped pedigree. A loop occurs when the graph has a *cycle* (in graph-theory terminology). A pedigree can also be represented as a connected graph with nodes of two types (see Figure 2): the nodes (or vertices) that represent the individuals (the v_i's), and the nodes of marriages (the m_i's). Here again, there are edges (the e_i's) that connect the nodes. In general, a graph G is represented under the form $G=(N,E)$ where N is the set of nodes and E is the set of edges of G. Note that each arc e_i of the second graph representation links an individual and a marriage node. In Figure 2, we can define a loop as a sequence of no duplicated *adjacent* (i.e. linked by an edge) nodes $n_1 \ldots n_k$, except for $n_1 = n_k$. It is always possible to obtain a connected unlooped pedigree by eliminating some edges from any given connected pedigree with loops (for instance, by eliminating the dotted edges in Figure 2). This

follows from a theorem of graph theory (Theorem 3, Chapter 3, in [13]), which states that every connected graph has a connected and unlooped sub-graph, called a *tree*. If further edges are eliminated, the resulting sub-graph is more general: it is an unlooped sub-graph which is not necessarily connected (a union of trees) called a *forest*. We shall show how searching for such a forest is equivalent to identifying a set of loop breakers.

For the pedigree represented in Figure 2, one possibility is to cut the loops by cloning individuals 3, 6 and 9, as shown in Figure 3. However, it seems more profitable rather to select individual 4 as loop breaker because she participates in two loops, and in this manner, when coupled for instance with individual 8, all loops are cut. In the first situation, we chose three individuals to cut all the loops, while in the second case only two individuals were needed. The optimal selection (even simply in terms of *number* of loop breakers) among all the individuals is not a straightforward task in general.

In practical situations, one has only partial genotype information about the individuals in the pedigree. The number of possible genotypes for a non-genotyped member of the pedigree may vary, depending on the genotypic information about its relatives. For example, in a bi-allelic context, an unknown-genotype individual born from an **AA** father and an **Aa** mother (both with known genotype), has only 2 possible genotypes: **AA** and **Aa**. The exclusion of incompatible genotypes is a way to reduce the range of the summations in the likelihood calculation.

A reasonable criterion to discriminate among the various combinations of individuals chosen as loop breakers can be defined with the aim of minimizing the computation time of the pedigree likelihood. Lange and Elston's algorithm computes the likelihood of a looped pedigree by integrating the likelihood (of an unlooped pedigree) conditionally to the loop-breaker's genotypes. The computational time of the likelihood calculation grows with the total number of possible genotypes that need to be considered for each loop breaker. Thus, based on this criterion, the best candidates for loop breakers are the individuals that participate in many loops and have a small number of possible genotypes. This criterion involves two quantities, which must be weighted appropriately in order to be able to evaluate whether a given proposed loop-breaker set is better than another. A common practice, for this purpose, consists in establishing for each individual i, a weight, f_i, which is proportional to the number, g_i, of possible genotypes for individual i, and inversely proportional to the number of

clones (or copies, including the original individual), k_i, which individual i will require if it is selected as loop breaker. For instance, Becker *et al.* [7] propose the weight $f_i = \frac{\log(g_i)}{k_i}$. Note that this is not the unique reasonable way to set the weights *fi's* (one may consider, for example, using $f_i = \frac{g_i}{k_i}$, etc.). We say that a set of loop breakers $i_1,\ldots,i_m$ is optimal if it cuts all the loops in the pedigree graph (when cloning individuals $i_1,\ldots,i_m$), and if its total weight, $\sum_{j=1}^{m} f_{i_j}$, is minimal (among all possible loop-breaker sets).

We now present a polynomial-time dynamic-programming algorithm, which is performed on an auxiliary graph, in order to find a low-cost loop-breaker set for the original pedigree.

2. Selecting a loop-breaker set

First, we build a (looped) sub-pedigree by eliminating, from the graph of the original (looped) pedigree (e.g. Figure 2) all the edges corresponding to founder individuals and to individuals without progeny (individuals that clearly do not participate in the loops). This corresponds to eliminating *leaves* in the pedigree graph (in graph terminology, a leaf is a node involved in only one edge) and their connecting edges. Further leaf elimination is recursively performed on the resulting sub-pedigree until one obtains a sub-pedigree graph which has no leaves. To illustrate, Figure 4 displays the sub-pedigree obtained from Figure 2 after removing the founder individuals v_1, v_2, v_5, and the individual without progeny v_{10} (no further leaf remains in this case). Secondly, this (looped) sub-pedigree is used to construct an auxiliary graph (Figure 5) as follows. A *star* (a sub-graph for which all edges have one common extremity) of virtual clones (copies) replaces each individual node v_i. More precisely, the original individual node v_i and its adjacent edges are replaced by a copy of v_i, $d(i)$-1 clones (extra copies) of v_i (where $d(i)$ is the *degree* of node v_i, i.e. the number of edges adjacent to it): $v_{i1},\ldots, v_{id(i)-1}$, new edges, each of which links v_i to one of its clones, and, finally, each of the original edges linking v_i (in the original sub-pedigree graph) to a node n is replaced (in the auxiliary graph) by an edge linking the same node n to one of the clones of v_i. Note that contracting these stars recovers the original sub-pedigree graph. Figure 5, illustrates

the auxiliary graph constructed from the sub-pedigree graph of Figure 4. The star s_3 introduced in the auxiliary graph for individual 3 contains only one edge c_{31}, while individual 4 has a 2-edge star s_4 formed by the edges c_{41} and c_{42}. The auxiliary graph constructed in this way, has the property that each *k*-edge star (dotted in Figure 5) represents a potential loop breaker.

Let us denote by E the set of all the edges of the auxiliary graph. The auxiliary graph involves two types edges in E: those which come directly from the original sub-pedigree graph (between an individual node and a marriage node), and the newly-introduced star edges (between a potential loop-breaker individual node, and each of its clone nodes). Let us call the edges of the first type (those coming from the original sub-pedigree graph), the *original edges* (the solid edges in the example of Figure 5). We denote by E_e the set of the original edges. From now on we shall call the above-constructed stars, the *proper stars* (they are s_3, s_4, s_6, s_7, s_8, s_9, dotted in Figure 5). We denote by S_s the set of proper stars, and by E_s the set of edges constituting the proper stars (so that we have $E = E_e \cup E_s$). In order to simplify the presentation of our algorithm, we shall consider that the original edges are special cases of stars (since they are 1-edge stars), so that we shall often refer to stars in general (original edges plus proper stars). We shall denote by S the set of all the stars (original edges plus proper stars), so that we have $S = E_e \cup S_s$) .

The algorithm we are about to describe follows from a simple observation. A forest in the auxiliary graph corresponds to a set of loop breakers (which cuts all the loops) for the original pedigree graph, as long as this forest is such that each proper star in S is either absent from it or present in its totality (in other terms, we do not consider selecting a strict subset of the edges of any given proper star). We shall call an *S-forest*, a subset of S constituted of all E_e plus a subset of S_s, and whose corresponding set of edges form a forest. For instance, Figure 6 displays such an *S*-forest (in fact, in this case the forest is even a tree). We shall say that an individual v is chosen as a loop breaker if the proper star associated with v is not in the proposed *S*-forest, and vice versa (if the star associated with an individual v is in the proposed *S*-forest, then v is not chosen as a loop breaker). Coming back to our example, the forest of Figure 6 corresponds to selecting v_3, v_6, and v_9 as loop breakers. Thus, if we assign to each proper star s_i, a weight $w_{s_i} := f_i$, where f_i is the (given) weight corresponding to individual i, then the problem of finding a minimal-cost (with respect to the given weights *fi*'s) loop-

breaker set reduces to finding in the auxiliary graph, a maximal-cost (with respect to the *w*'s) *S*-forest, *F,* i.e. a maximal-cost forest made with all the original edges plus some of the proper stars. In the example of Figure 5, *F* may be equal to *S* - {s_3, s_6, s_9} or equal to *S* - {s_4, s_8} depending on the given weights *fi*'s. For instance, if one wishes to use the weights $f_i = \frac{\log(g_i)}{k_i}$ proposed by Becker *et al.* in [7], then one can simply set $w_{s_i} := \log\frac{(g_i)}{k_i}$ for any proper star.

The problem of finding a minimal loop-breaker set is known in graph theory (feedback vertex set) to be NP-complete (see [14]). In other words, specialists do not expect that a polynomial-time algorithm could exist to solve it optimally.

Given the above-defined weights *w*'s associated with every proper star of our auxiliary graph, then the problem of selecting a minimal (with respect to the f_i's) loop-breaker set is equivalent to finding in *S*, a minimal-cost set of proper stars, *S**, whose removal from the auxiliary graph gives a forest *F* (i.e. an unlooped sub-graph). The selected set of proper stars *S** then corresponds to the optimal loop-breaker set.

Here is the dynamic-programming algorithm SFH (S-Forest Heuristic) that we propose in order to find a good (high cost) S-forest, $\overline{F}$, which in turn corresponds to a good (low cost) loop-breaker set. At any given step *j*, it finds a high-weight S-forest built with all the original edges, plus a subset of the proper stars constituted of a total of *j* edges. Let us first set the notation:

$d(i)$: The degree of individual v_i in the original sub-pedigree graph;

$|s_i|$: the number of edges in star s_i (= $d(i)$-1);

$|F|$: the total number of edges in the stars constituting S-forest *F*;

$|E_s|$: the total number of edges in the proper stars ($=\sum_i (d(i)-1)$, summing over all individuals of the original sub-pedigree);

K: the maximal possible number of edges for a star in the problem under consideration (=maximal degree over all individual nodes in the original sub-pedigree graph, minus 1).

Furthermore, we keep in memory for each step j=0, 1, 2,…, $|E_s|$ the following:

F_j : the highest-cost S-forest F that could be found such that $|F| = |E_e| + j$;
f_j : the cost of F_j (= the total weight of the proper stars in F);

$\overline{s}_j$: the star entering F_j at step j.

The SFH algorithm

Step 0 : $F_0 := |E_e|; f_0 := 0$
For j from 1 to $|E_s|$ do:

Step j : Let $\overline{K} = \min(j, K)$, and

$$f_j := \max \begin{Bmatrix} f_{j-1} + \max\{w_s \text{ such that } |s| = 1,\ s \in S_s - F_{j-1}, F_{j-1} \cup \{s\} \text{ has no cycle, and } f_{j-1} \neq -1\} \\ f_{j-2} + \max\{w_s \text{ such that } |s| = 2,\ s \in S_s - F_{j-2}, F_{j-2} \cup \{s\} \text{ has no cycle, and } f_{j-2} \neq -1\} \\ \text{M} \\ f_{j-\overline{K}} + \max\{w_s \text{ such that } |s| = \overline{K},\ s \in S_s - F_{j-\overline{K}}, F_{j-\overline{K}} \cup \{s\} \text{ has no cycle, and } f_{j-\overline{K}} \neq -1\} \end{Bmatrix} \quad (1)$$

If none of the inner maximizations in (1) involves candidates, then simply set $f_j := 0, F_j := \varnothing$; and go to step j+1. Otherwise: let $\overline{s}$ be the proper star which determines the maximum in (1), and let $\overline{k} := |w_{\overline{s}}|$.

Set, $f_j := f_{j-\overline{k}} + w_{\overline{s}}$,
$F_j := F_{j-\overline{k}} \cup \{\overline{s}\}$, and $\overline{s}_j := \overline{s}$.

Step $|E_s| + 1$: The cost of a good S-forest is $f^* := \max_{j=1,2,..|E_s|} \{f_j\}$.
Let $j_1 := \underset{j=1,2,..|E_s|}{\operatorname{argmax}} \{f_j\}$ (i.e. such that $f_{j_1} = f^*$), $j_2 := j_1 - |\overline{s}_{j_1}|$, $j_3 := j_2 - |\overline{s}_{j_2}|$,…, until obtained an n such that $j_n - |\overline{s}_{j_n}| = 0$. Set $\overline{F} := \{\overline{s}_{j_1}, \overline{s}_{j_2}, ..., \overline{s}_{j_n}\}$.
The loop breakers we propose are the individuals corresponding to the proper stars in $S_s - \overline{F}$

For each step of the SFH algorithm, there is a memory requirement of one real value (f_j), one integer (the index of the star entering F_j, and a set of the indices of the stars in F_j. The algorithm requires initially to sort in decreasing order all 1-edge proper stars, all 2-edge proper stars, …, and all K-edge proper stars. Moreover, the computational work of this algorithm involves performing $|E_s|$ times Step j. Each such step involves performing the

maximization (1), *i.e. K* additions, followed by the simple comparison of *K* values. The algorithm SFH is inspired from Kruskal algorithm to find a maximal-cost spanning tree in a graph (see for instance [15]), combined with the basic principle of dynamic programming [16].

To conclude this presentation of our new algorithm SFH , here is an implementation detail which will help the reader in programming our algorithm. In analogy with the classical Kruskal algorithm for finding an optimal spanning tree in a graph $G=(N,E)$, here is how we verify whether a given sub-graph G' has no cycle (this is needed when performing the maximizations in (1)). At each iteration, the sub-graph G' we consider is made of all the nodes of the original graph G, together with a subset, E', of the edges of G. At a particular step j of our algorithm, E' consists in the edges forming F_j. At each step of the SFH algorithm, we update the number, C, of connected components of G'. We can then easily use the classical result of graph theory which stipulates that G' has no cycle if and only if $|N|-|E'|\leq|C|$.

Computational results

Our algorithm was implemented in Fortran 95, and the programs were run on a RISC 6000 NH2 (PowerPC3). In order to demonstrate the performance of the algorithm, we simulated three looped pedigrees with multiples marriages. In all pedigrees, the proportion of males and females were 33% and 67%, respectively. In each pedigree, a base population (unrelated individuals) was simulated for the first year, and mating and replacement through the successive years built the rest of the pedigree. The replacement rate was 50% for both sexes, and for each year. Individuals for replacement were chosen at random. Matings among parents were also performed randomly. The main features of these test problems are described in Table 1.

In order to allow a fair comparison, we used the same criterion (the same f_i's) as that implemented in the software package FASTLINK 4.1P in order to evaluate the value (the cost) of any proposed set of loop breakers. More precisely, the cost of a given set of loop breakers was calculated using the sum of the functions defined in Becker *et al.* [7] over the loop breakers: $f_i = \frac{\log(g_i)}{k_i}$, where g_i is the number of possible genotypes for individual *i,*

and k_i is the number of copies required for individual i (it is also the degree of i in the original pedigree graph, or number of marriages in which individual i is involved, including the marriage of the parents of i). The genotype elimination algorithm presented in [17] was used to find g_i, taking into account, at each step, the compatibility among the genotype information from parents, mates and progeny. The numerical results are reported in Table 2.

For Pedigree I, both approaches selected four individuals as loop breakers, two of them with three possible genotypes. The other selected individuals had known genotype for the result proposed by SFH, while FASTLINK 4.1P preferred one individual with known genotype and another with two possible genotypes. The loop-breaker sets found by the methods had different cost. The method we are introducing in the current paper finds a solution whose cost is much lower (by almost 24%) than that produced by FASTLINK 4.1P, and within a total CPU time which is five times inferior. When ran on Pedigree II, the methods found this time even a different number of loop breakers. The cost of our solution is again better (by 12%), and it is found in ten times less CPU time. For the largest test problem (Pedigree III), we could not achieved to have FASTLINK 4.1P to terminate, while our approach still computes a low-cost loop-breaker set.

Conclusion

Extensions of the Elston-Stewart algorithm have been developed for evaluating the likelihood without selecting loops breakers [3, 4]. Another approach is to cut loops, and to generate an unlooped pedigree in which Lange and Elston's conditioning method can be used. For this purpose, various heuristic methods have been proposed to find reasonable loop-breaker sets. However, attempting at finding better loop-breaker set is largely motivated, due to the complexity of conditional likelihood computation.

In this paper, we introduced a polynomial-time method, which proposes a low-cost loop-breaker set in general pedigrees (multiples marriages). It compares very favourably with the corresponding dedicated program in the software package FASTLINK 4.1P, in terms of the quality of the solution obtained, in terms of CPU time, and in terms of the size of the problem that can be addressed.

Acknowledgements

The authors wish to thank Dr. Miguel Peréz-Enciso, for helpful comments. ZV is funded by Ministerio de Educación, Ciencia y Técnica de la República Argentina and INRA.

References

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

2 Lange K, Elston RC: Extentions to pedigree analysis. I. Likelihood calculations for simple and complex pedigrees. Hum Hered 1975; 25: 95-105.

3 Cannings C, Thompson EA, Skolnick EH: Probability functions on complex pedigrees. Adv Appl Prob 1978; 10: 26-61.

4 Lange K, Boehnke M: Extensions to pedigree analysis. V. Optimal calculation of Mendelian Likelihoods. Hum Hered 1986, 33: 291-301.

5 Wang T, Fernando RL, Stricker C, Elston RC: An approximation to the likelihood for a pedigree with loops. Theor Appl Genet 1996; 93:1299-1309.

6 Xie X, Ott J: Finding all loops in a pedigree. Am J Hum Genet 1992; 51: A205.

7 Becker A, Geiger D, Schäffer AA: Automatic selection of loop breakers for genetic linkage analysis. Hum Hered 1998; 48: 49-60.

8 Becker A, Bar-Yehuda R, Geiger D: Random algorithms for the loop cutset problem. Journal of Artificial Intelligence Research 2000; 12: 219-234.

9 Stricker C, Fernando RL, Elston RC: An algorithm to approximate the likelihood for pedigree data with loops by cutting. Theor Appl Genet 1995; 91:1054-1063.

10 O'Connell JR, Weeks DE: An optimal algorithm for automatic genotype elimination. Am J Hum Genet 1999; 65: 1733-1740.

11 Cottingham Jr. RW, Idury RM, Schaffer AA: Faster sequential genetic linkage computations. Am J Hum Genet 1993; 53: 252-263.

12 Schaffer AA, Gupta SK, Shriram K, Cottingham Jr. RW: Avoiding recomputation in linkage analysis. Hum Hered 1994; 44: 225-237.

13 Berge C: Graphes. Gauthier-Villars, Paris, 1970.

14 Festa P, Pardalos PM, Resende MGC: Feedback set problems. In C Floudas and PM.Pardalos (eds), Encyclopedia of optimization, Kluwer Academic Press 2001; vol 2, pp. 94-106.

15 Charon I, Germa A, Hudry O : Méthodes d'optimisation combinatoire. Masson, Paris, 1996.

16 Bradley S P, Hax A C, Magnanti T L: Applied Mathematical Programming. Addison-Wesley, 1977.

17 Lange K, Goradia TM: An algorithm for automatic genotype elimination. Am J Hum Genet 1987; 40: 250-256.

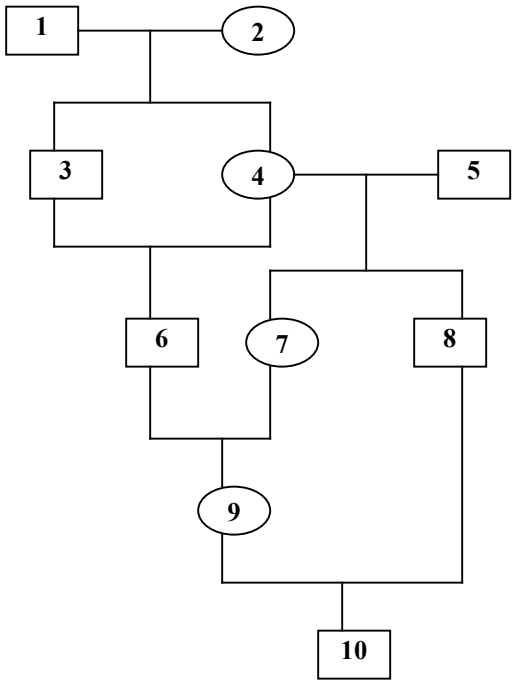

Figure 1. Pedigree with loops

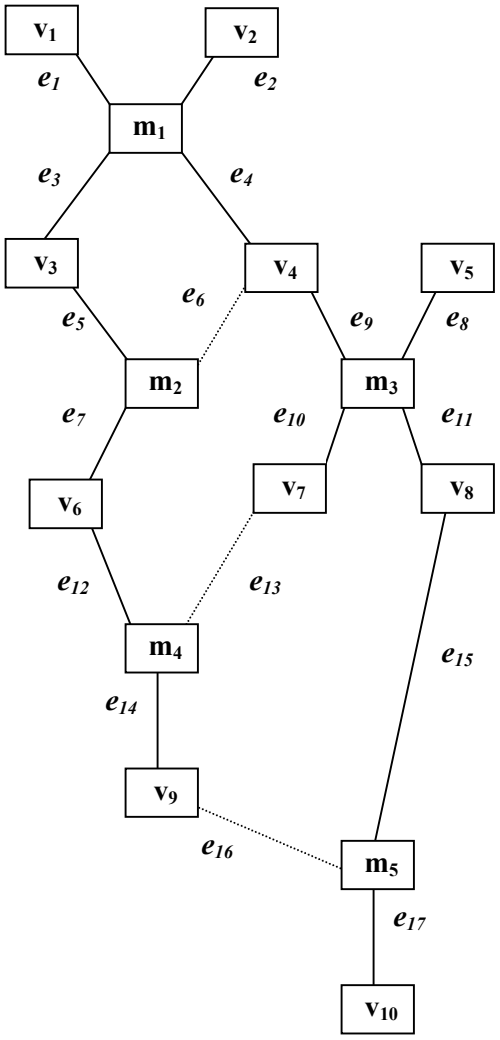

Figure 2. Graph of a looped pedigree

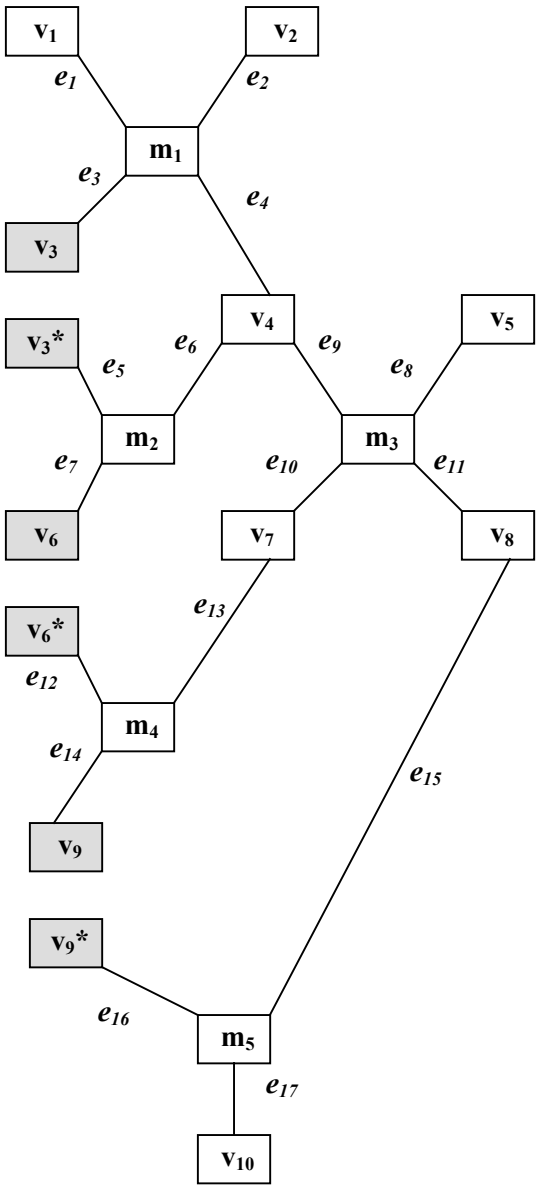

Figure 3. Pedigree graph after "cloning"

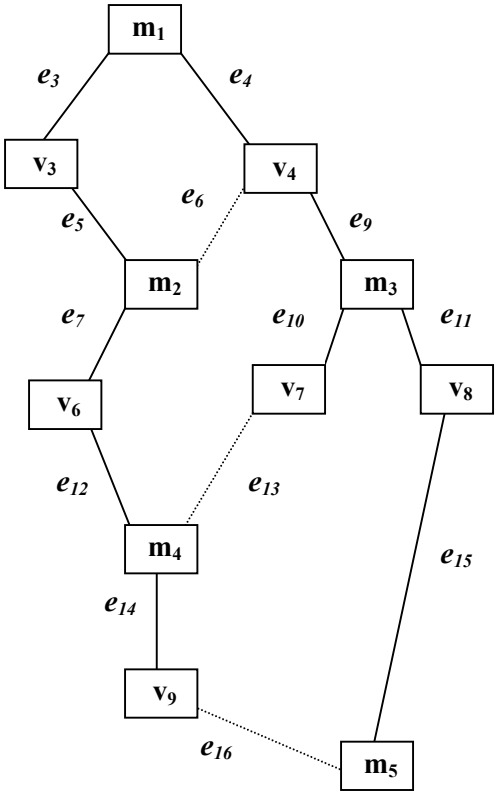

Figure 4. Sub-pedigree graph

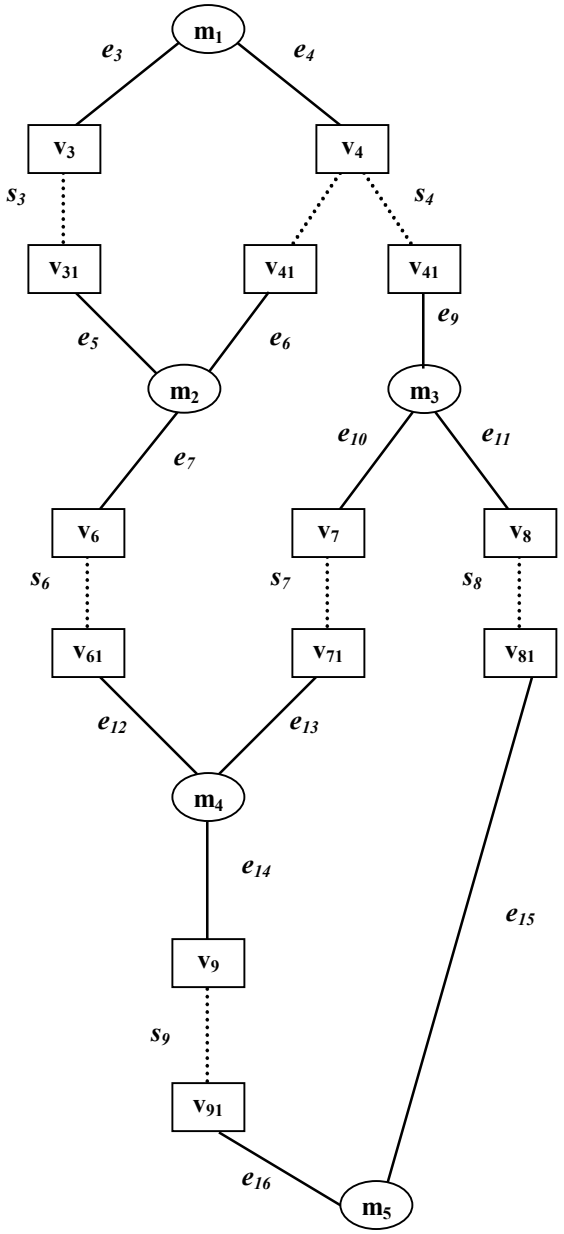

Figure 5. The auxiliary graph and its stars (dotted edges)

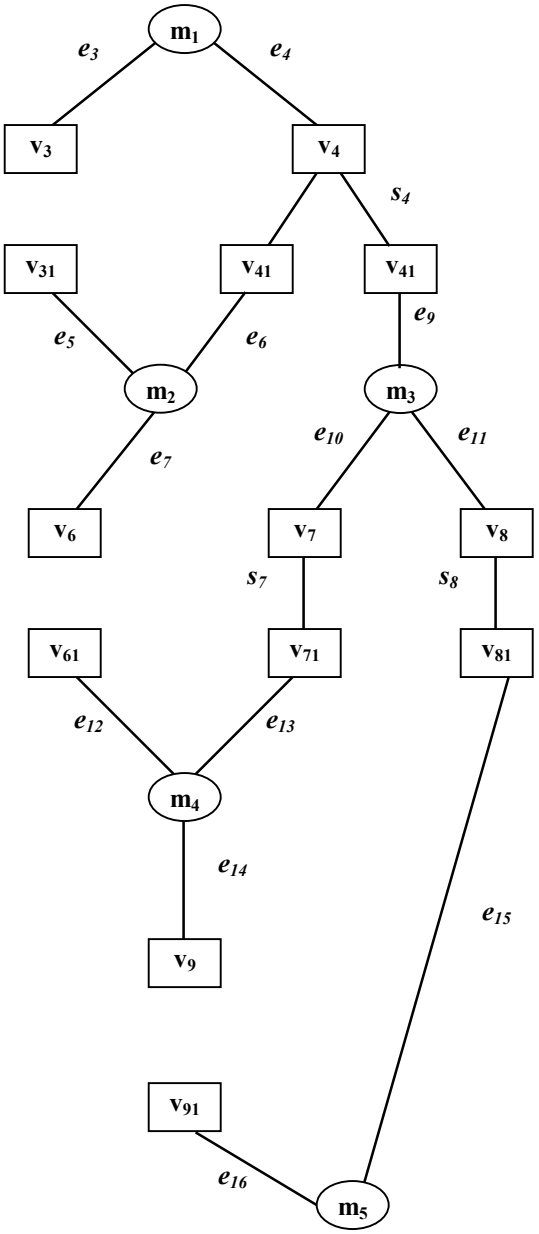

Figure 6. The graph of a forest

Table 1. Test problems: three simulated looped pedigrees

Pedigree	Total number of individuals	Number of individuals in loops	Number of loops	Possible number of marriages per individual	K
I	60	15	7	1, 2	2
II	350	137	41	1,2	2
III	1000	617	361	1,2	2

Table 2. Computational comparison: cost, number of loop breakers, and CPU time

Pedigree	FASTLINK			SFH		
	Cost	Number of Loop Breakers	CPU Time (s.)	Cost	Number of Loop Breakers	CPU Time (s.)
I	0.9634	4	0.41	0.7324	4	0.08
II	7.0537	21	103.47	6.1860	23	10.94
III				58.5247	189	619.01

Figure 1. Pedigree with loops.

Figure 2. Graph of a looped pedigree.

Figure 3. Pedigree graph after “cloning”.

Figure 4. Sub-pedigree graph.

Figure 5. The auxiliary graph and its stars (dotted edges).

Figure 6. The graph of a forest.

Table 1. Test problems: three simulated looped pedigrees.

Table 2. Computational comparison: cost, number of loop breaker, and CPU time.