Am. J. Hum. Genet. 65:1733–1740, 1999

An Optimal Algorithm for Automatic Genotype Elimination

Jeffrey R. O'Connell[1,2] and Daniel E. Weeks[1]

[1]Department of Human Genetics, University of Pittsburgh, Pittsburgh, and [2]The Wellcome Trust Centre for Human Genetics, University of Oxford, Windmill Road, Oxford

Summary

In an effort to accelerate likelihood computations on pedigrees, Lange and Goradia defined a genotype-elimination algorithm that aims to identify those genotypes that need not be considered during the likelihood computation. For pedigrees without loops, they showed that their algorithm was optimal, in the sense that it identified all genotypes that lead to a Mendelian inconsistency. Their algorithm, however, is not optimal for pedigrees with loops, which continue to pose daunting computational challenges. We present here a simple extension of the Lange-Goradia algorithm that we prove is optimal on pedigrees with loops, and we give examples of how our new algorithm can be used to detect genotyping errors. We also introduce a more efficient and faster algorithm for carrying out the fundamental step in the Lange-Goradia algorithm—namely, genotype elimination within a nuclear family. Finally, we improve a common algorithm for computing the likelihood of a pedigree with multiple loops. This algorithm breaks each loop by duplicating a person in that loop and then carrying out a separate likelihood calculation for each vector of possible genotypes of the loop breakers. This algorithm, however, does unnecessary computations when the loop-breaker vector is inconsistent. In this paper we present a new recursive loop breaker–elimination algorithm that solves this problem and illustrate its effectiveness on a pedigree with six loops.

Introduction

Many pedigree-based statistics in the area of linkage and segregation analysis are based on computing the likelihoods of pedigree data. These likelihoods often can be quite difficult to compute, since (a) the computation involves summation over every possible underlying combination of multilocus genotypes that is consistent with the observed phenotypic information and (b) the computational time of likelihood calculation grows exponentially with the number of genotypes. In an effort to accelerate likelihood computations, Lange and Goradia (1987) defined a genotype-elimination algorithm that aims, on a locus-by-locus basis, to identify those genotypes that are not consistent with the observed phenotype information in the pedigree and that, because they contribute no information, therefore can be eliminated from the likelihood computation. Although the Lange-Goradia algorithm is in common use today and is quite effective in speeding up likelihood computations, it is not optimal for those pedigrees which continue to pose daunting computational challenges—namely, pedigrees with loops in them. We propose here a simple extension of the Lange-Goradia algorithm that we prove is optimal for pedigrees with loops, and we give an example of its use in detection of genotyping errors. We also introduce a more efficient method for performing genotype elimination within a nuclear family, the fundamental operation in genotype elimination for the entire pedigree. Finally, we introduce a recursive genotype-elimination algorithm to determine all consistent loop-breaker vectors in a pedigree with loops.

Received June 7, 1999; accepted for publication September 22, 1999; electronically published November 2, 1999.

Address for correspondence and reprints: Dr. Jeff O'Connell, University of Pittsburgh, Department of Human Genetics, Crabtree Hall A310, 130 DeSoto Street, Pittsburgh, PA 15261. E-mail: jeff@watson.hgen.pitt.edu

0002-9297/1999/6506-0030$02.00

An Optimal Genotype-Elimination Algorithm

We first define some necessary terms. A pedigree may be represented as a directed graph, with two types of vertices: *person vertices* and *mating vertices*. The arcs of such a graph fall into two classes: *mating arcs,* which connect two parents to a mating, and *descent arcs,* which connect a mating to the children produced by that mating. Note that each arc always contains a mating vertex and a person vertex. For example, in figure 1, the arc from person 1 to mating vertex M1 is a mating arc, whereas the arc from M1 to person 4 is a descent arc. A *loop* is a sequence of arcs that starts and ends in the same vertex; note that although the definition of a loop ignores the directionality of the arcs, an arc may appear only once in a given loop. In figure 1, the loop consists of the arcs M1-3, 3-M2, M2-4, and 4-M1. We restrict our consideration to pedigrees that are graphically *con-*

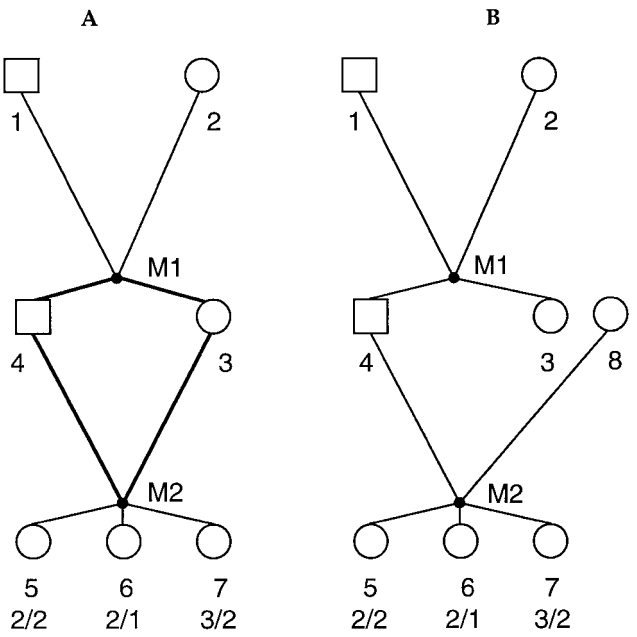

Figure 1 *A*, Pedigree (from Sobel et al. 1995) that has a loop (indicated by the thicker lines), since parents 3 and 4 are siblings. *B*, Broken pedigree. To break the loop using person 3, we would duplicate her and assign her duplicate, person 8, the mating arc that formerly went from person 3 to the mating vertex M2.

nected—that is, in which every person vertex can be reached (ignoring directionality), by some sequence of distinct arcs, from any other person vertex.

Likelihood computations on a pedigree with loops are often facilitated by "breaking the loops" via the judicious selection of loop breakers (Lange and Elston 1975). One breaks a loop by selecting a person in the loop, detaching all mating arcs from him or her, and reattaching them to a new dummy individual, the *loop breaker,* who, during the computation of the likelihood, is constrained to always have the same genotype as the original person. A *loop-breaker vector* is a vector containing one genotype for each loop breaker. When computing the overall likelihood of the pedigree, we must compute the likelihood for each possible loop-breaker vector and then sum these results to obtain the total likelihood. We say that a loop-breaker vector is *inconsistent* if its genotypes and the observed phenotypes of the rest of the pedigree are mutually impossible under Mendelian laws of inheritance but is *consistent* otherwise. In figure 1, there are two choices for a loop breaker: individuals 3 and 4. Because of symmetry, either loop breaker will have four loop-breaker vectors: (1|2), (2|1), (3|2), and (2|3), where 1|2 represents the ordered genotype with maternal allele 1 and paternal allele 2, etc. With this loop-breaker approach, a connected pedigree with loops can always be broken to form a new connected pedigree without loops, provided that we permit a person to break more than one loop at a time. (This result follows from a basic theorem, from graph theory, that states that every connected graph has a spanning tree, which is connected and has no loops. A spanning tree may be found by repeatedly choosing a loop and removing one of its arcs until there are no loops left. The person vertex of each arc that has been removed represents a loop breaker.) Note that, although the number of loops in a pedigree is an invariant of the pedigree, when we permit a person to break more than one loop, the number of loop breakers need not be an invariant. Becker et al. (1998) discuss how to choose loop breakers efficiently to minimize the number of loop-breaker vectors required by the linkage-analysis program FASTLINK (Lathrop and Lalouel 1988; Cottingham et al. 1993).

Given a connected pedigree with n persons, a genotype vector $(G_1, G_2,\ldots, G_n)$, where G_i is the genotype of the ith person, is *compatible* if it is consistent with the observed phenotypes of the pedigree members. (Note that, for codominant markers, the observed phenotype is simply the observed genotype.) Genotype-elimination algorithms typically aim to construct, for each person, a minimal genotype list that contains only genotypes that are the members of at least one compatible genotype vector. If a certain genotype is never a member of any compatible genotype vector, then it is *superfluous*. If the genotype-elimination algorithm is optimal, then it eliminates every superfluous genotype from each person's genotype list.

Lange and Goradia (1987) proposed an improved genotype-elimination algorithm based on a previous algorithm developed by Lange and Boehnke (1983). They proved that their new algorithm was guaranteed to eliminate all superfluous genotypes on any connected pedigree without loops. However, they showed by counterexample that their algorithm was not optimal, because it could fail to eliminate all superfluous genotypes from pedigrees with loops. As Lange and Goradia (1987) did, we restrict our attention to genotype elimination at a single locus at a time. Note that genotype elimination can be used in a multilocus context to eliminate both superfluous genotypes and superfluous phases, as discussed by Lange and Weeks (1989). Since our optimal algorithm is based on the Lange-Goradia algorithm, we repeat that algorithm verbatim from the original publication (Lange and Goradia 1987).

The Lange-Goradia Genotype-Elimination Algorithm

A. For each pedigree member, list only those genotypes compatible with his or her phenotype.

Table 1

Ordered Genotype Lists for the Pedigree in Figure 1

Person	Genotype List[a]	
	After Lange-Goradia Algorithm	After Optimal Algorithm
1	{**1**|**1**, 2|1, 3|1, 1|2, 2|2, 3|2, 1|3, 2|3, **3**|**3**}	{2|1, 3|1, 1|2, 2|2, 3|2, 1|3, 2|3}
2	{**1**|**1**, 2|1, 3|1, 1|2, 2|2, 3|2, 1|3, 2|3, **3**|**3**}	{2|1, 3|1, 1|2, 2|2, 3|2, 1|3, 2|3}
3	{2|1, 1|2, 3|2, 2|3}	{2|1, 1|2, 3|2, 2|3}
4	{2|1, 1|2, 3|2, 2|3}	{2|1, 1|2, 3|2, 2|3}
5	{2|2}	{2|2}
6	{2|1, 1|2}	{2|1, 1|2}
7	{3|2, 2|3}	{3|2, 2|3}

[a] Superfluous genotypes are indicated in bold. An ordered genotype has the maternal allele listed first, whereas an unordered genotype does not distinguish maternal or paternal origin. For example, if we were using unordered genotypes, person 6 would have the genotype list {1/2}.

B. For each nuclear family:

1. Consider each mother-father genotype pair.
   a. Determine which zygote genotypes can result.
   b. If each child in the nuclear family has one or more of these zygote genotypes among his current list of genotypes, then save the parental genotypes. Also save any child genotype matching one of the created zygote genotypes.
   c. If any child has none of these zygote genotypes among his current list of genotypes—that is, if he is incompatible with the current parental pair of genotypes—take no action to save any genotypes.
2. For each person in the nuclear family, exclude any genotypes not saved during step 1 above.

C. Repeat part B until no more genotypes can be excluded.

We now extend the Lange-Goradia algorithm to be optimal in the presence of loops, by using loop-breaker vectors to transform the genotype-elimination problem into a collection of problems involving loopless pedigrees. Schäffer (1996) uses the technique of loop-breaker vectors to eliminate inconsistent loop-breaker vectors in FASTLINK (Cottingham et al. 1993), but he does not explicitly use the Lange-Goradia algorithm for genotype elimination, nor does he address optimality.

Our Optimal Genotype-Elimination Algorithm

A. Do conventional genotype elimination on the pedigree (without breaking the loops) to reduce the size of each person's genotype list.

B. Break the loops and construct the set of loop-breaker vectors. For each loop-breaker vector, assign the loop breakers their respective genotypes from the vector and carry out genotype elimination on the broken pedigree structure. If the loop-breaker vector is consistent, then retain the genotypes for each person remaining after elimination.

C. For each person, construct his or her final genotype list by taking the union of the retained genotype lists.

Theorem: The algorithm above eliminates all superfluous genotypes.

Proof: In step B, after breaking the loops, we now have a connected pedigree without loops. Thus, after genotype elimination, if the loop-breaker vector is consistent, the pedigree is guaranteed to contain no superfluous genotypes, as proved by Lange and Goradia (1987). Taking the union (step C) of sets containing no superfluous genotypes will not introduce any new superfluous genotypes, so the final union contains no superfluous genotypes.

Illustration

Sobel et al. (1995) used the inbred pedigree in figure 1 to illustrate the failure of the Lange-Goradia genotype-elimination algorithm to eliminate all superfluous genotypes. As they point out, the Lange-Goradia algorithm fails to eliminate the superfluous genotypes 1/1 and 3/3 from the genotype lists for persons 1 and 2 (table 1). It fails because it does not take into account the symmetry condition that, whenever person 3 has the unordered genotype 1/2, person 4 must simultaneously have the unordered genotype 2/3, or vice versa. However, this symmetry is captured by our optimal algorithm, since the algorithm steps through each of the possible two genotypes (1/2 and 2/3) for person 3 (and her duplicate 8 in fig. 1*B*). When persons 3 and 8 are simultaneously assigned the genotype 1/2, then genotype elimination on the loopless pedigree in figure 1*B* will eliminate genotypes 1/2 and 2/1 from person 4's genotype list, thus

capturing the symmetry. Similarly, for person 3's second loop-breaker vector containing 2/3, the genotypes 2/3 and 3/2 are eliminated from person 4's genotype list. Thus, our optimal genotype-elimination algorithm is successful (as expected) in eliminating genotypes 1/1 and 3/3 from the genotype lists for persons 1 and 2 (table 1).

Complexity of the Algorithms

Two important aspects of any algorithm are its time complexity and its space complexity. To determine the space complexity of the Lange-Goradia and our genotype-elimination algorithm for a given locus, let I be the number of individuals in the pedigree and let N be the number of alleles at the locus. Then the space complexity is bounded by $I*N*N$, where $N*N$ represents the maximum number of ordered genotypes of any untyped individual. In general, storage space is not a problem. To determine the time complexity of our algorithm, first let the time complexity of the loopless Lange-Goradia algorithm be T. Then, since our algorithm proceeds by constructing the set of loop-breaker vectors from all possible combinations of genotypes taken from the genotype lists of the loop breakers after Lange-Goradia genotype elimination, we see that our algorithm is multiplicative in the number of loop breakers, say L. So the total time complexity is bounded by $L*T$. For example, if we had a pedigree with two loop breakers, and if their genotype lists contained 10 and 20 genotypes after Lange-Goradia elimination, then we would have to consider 200 loop-breaker vectors. Obviously, as the number of loop breakers increases, the number of loop-breaker vectors may increase dramatically. There are two ways in which we can speed up our algorithm: speeding up the original Lange-Goradia algorithm and decreasing the number of loop breakers.

Sequential Genotype Elimination

For each nuclear family, the Lange-Goradia algorithm considers all mother-father genotypes and determines whether these two genotypes and the genotypes of any children are to be saved. This requires on the order of $M*F*A$ parent-child genotype checks, where M and F are the number of genotypes of the mother and father, respectively, and A is the sum of the number of genotypes of all the children. This method of checking, however, may produce many redundant checks, because, for example, a single parent genotype may be saved as part of checking many mother-father genotype pairs, even if it has already been saved from a previous check. A method of avoiding this redundancy is checking each individual in the nuclear family separately, to determine whether that person's genotypes are either compatible with the genotypes of the other individuals, and thus saved, or incompatible, and thus not saved. In addition, for each genotype of that individual that is saved, we also save the genotypes of the other individuals in the nuclear family that were found to be compatible during this particular check. For example, we start by marking all genotypes in the nuclear family as not saved, and then we consider the first individual, say, the mother. Given her genotype G_m, we check whether G_m is compatible with the father and children, by searching for the first possible configuration of the father and children that is compatible with G_m. If such a configuration is found, we save each genotype in the configuration; otherwise, we delete G_m from the mother's genotype list. Note that we can save genotypes of each person in the nuclear family as soon as we know that they are mutually compatible, but we can delete genotypes from one specific person only after checking all possibilities. To increase the chance of finding a different father-children genotype configuration to save when checking the next genotype of the mother, we use a shifting technique, as follows. When we save a genotype, we shift it to the end of the person's genotype list, thereby ensuring that we always consider as-yet-unsaved genotypes before already-saved ones. Ideally, if the mother has n saved genotypes, we will also save n different father-children genotype configurations. After checking the mother, we check the father. But now, instead of having to check *every* genotype of the father, we only need to check those genotypes that are still not saved. We proceed in the same way, looking for the first mother-children configuration, if it exists, and, if it does, saving the genotypes of the children (the mother's genotypes do not need to be saved again). This procedure continues until each individual in the family has been checked. We outline this algorithm in more detail below. Since it requires more checks to determine whether we do not save a genotype than whether we do save it, we also add a useful preprocessing step that does a parent-child check with only two individuals at a time (instead of both parents and all children) when either the child or parent is typed. For example, if a child is typed 3/4, and the mother is untyped, we can eliminate all of the mother's genotypes that have neither the 3 nor the 4 allele, regardless of the genotypic information of the father and the other children.

The Sequential Genotype-Elimination Algorithm

A. For each pedigree member, list only those genotypes compatible with his or her phenotype.

B. For each nuclear family:

   1. For each parent, determine whether an untyped parent is compatible with any typed children separately. For each untyped child, determine

whether the child is compatible with each typed parent separately. Delete any incompatible genotypes.

2. Mark all genotypes of the nuclear family as unsaved.
3. For each genotype of the mother:
   a. Form a mother-father genotype pair, using the next genotype in the genotype list of the father.
   b. Determine which zygote genotypes can result.
   c. If each child has at least one of these zygote genotypes among its list of genotypes, save the mother, father, and children genotypes and, for each individual, move the newly marked saved genotypes to the end of its respective genotype list.
   d. If any child has none of the zygote genotypes among its current list of genotypes, repeat step a.
4. Delete any of the mother's genotypes not marked saved.
5. Repeat part c, switching the roles of the mother and father but checking only the genotypes of the father that are still marked unsaved.
6. For each child:
   a. For each genotype marked unsaved, determine the mother-father genotype pair compatible with that genotype.
   b. Determine if any mother-father genotype pair formed in step 6.a. is compatible with the other children. If so, save the genotypes of the children. If not, delete the unsaved genotype from the child's genotype list.

C. Repeat part B until no more genotypes can be excluded.

Whereas the original Lange-Goradia algorithm *always* has order $M*F*A$ checks, the best case for our sequential algorithm would be that, after checking the mother, we would not have to check the father and children, thus reducing the number of checks from order $M*F*A$ to order $M*(F+A)$. The efficiency of this algorithm increases as more incompatible genotypes are deleted from the pedigree during each round of genotype elimination.

Recursive Genotype Elimination

We can also speed up our algorithm by applying similar ideas in order to avoid redundant checking of inconsistent loop-breaker vectors. As we saw above, the number of loop-breaker vectors is multiplicative in the number of loops and can grow very quickly. For example, the pedigree in figure 2 has six loops (one choice of loop breakers—22, 26, 15, 14, 12, and 9—is designated by the solid individuals) and 409,600 loop-breaker vectors at the given marker. Performing genotype elimination 409,600 times for each loop-breaker vector took

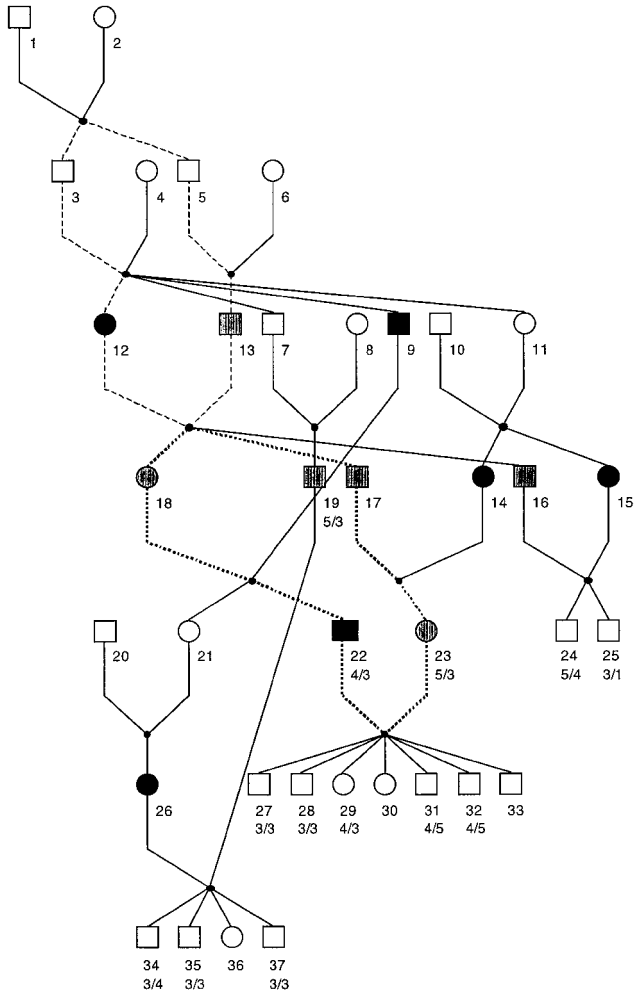

Figure 2 Human pedigree from Saudi Arabia, with six loops (Jones et al. 1998). Two of these loops are indicated, by the dashed and dotted lines. The following output from the LOOPS program (Xie and Ott 1992) indicates the sequence of individuals and marriage nodes (in parentheses) that make up the loop. Loop 1, 7-(7,8)-19-(19,26)-26-(20,21)-21-(18,9)-9-(3,4)-7; Loop 2, 7-(7,8)-19-(19,26)-26-(20,21)-21-(9,18)-18-(13,12)-12-(3,4)-7; Loop 3, 3-(3,4)-7-(7,8)-19-(19,26)-26-(20,21)-21-(9,18)-18-(12,13)-13-(6,5)-5-(1,2)-3; Loop 4, 7-(7,8)-19-(19,26)-26-(20,21)-21-(9,18)-18-(12,13)-16-(16,15)-15-(10,11)-11-(3,4)-7; Loop 5, 16-(16,15)-15-(10,11)-14-(14,17)-17-(12,13)-16; and Loop 6, 18-(12,13)-16-(16,15)-15-(10,11)-14-(14,17)-23-(23,22)-22-(9,18)-18. Our set of loop breakers is indicated by the blackened symbols, whereas the gray symbols indicate the set of loop breakers selected by the algorithm of Becker et al. (1998).

320 min on our SUN Enterprise with a 300-MHz processor. The number of loop-breaker vectors for this pedigree, however, can be reduced, by a factor of 8.3, to 49,152, by use of set-recoded genotypes introduced by O'Connell and Weeks (1995), and the time reduced, by a factor of 17.7, to 18 minutes. (The set-recoding also reduces the genotype lists of other untyped individuals who are not loop breakers, which accounts for the speed-up >8.3.) However, of these 49,152 loop-breaker vectors, 14,270 vectors are consistent and 34,882 inconsistent with the pedigree data. We can avoid checking these 34,882 inconsistent loop-breaker vectors by recursively fixing each component of the vector and using conventional genotype elimination to reduce the genotype lists of the remaining loop breakers. For example, in figure 2 loop breaker 22 has two ordered genotypes: 4|3 and 3|4. After genotype elimination, person 9 (person 22's father) has eight genotypes, each containing either the 3 or the 4 allele. If we perform genotype elimination by using just the genotype 4|3 for person 22, the number of loop-breaker vectors is 15,360, and, if we use just the genotype 3|4, the number of loop-breaker vectors is also 15,360. Thus, if we perform genotype elimination twice, using each genotype once, the total number of loop-breaker vectors is reduced from 49,152 to 30,720. The reason for the reduction is that, when individual 22 is assigned the ordered genotype 4|3, all of individual 9's genotypes must have at least one 3 allele, thus reducing from eight to five the number of consistent genotypes that individual 9 has. In other words, we have avoided checking all the inconsistent loop-breaker vectors containing 4|3 for individual 22 and any one of those three inconsistent genotypes for individual 9, without having to consider which genotypes the remaining four loop breakers have. Now consider the next loop breaker, individual 26, who also has two possible genotypes: 3|4 and 4|3. By performing genotype elimination four times, once for each choice of genotypes for loop breakers 22 and 26, the number of loop-breaker vectors is further reduced, to 24,576. We can reduce this number further still, to 14,270 (thus avoiding all inconsistent loop-breaker vectors) by continuing in this manner with the remaining four loop breakers. The time required to perform the full recursive loop-breaker elimination was 90 s. We now present the algorithm:

Recursive Loop-Breaker Elimination Algorithm

Let N be number of loop breakers and let L_K be the genotype list of loop breaker K. Then the recursive procedure is:

Elimination (N, L_N)

A. If $N = 0$, return.

B. While L_N is not empty:

1. Choose G in L_N and fix loop breaker N.
2. Perform conventional Lange-Goradia elimination to find L_N-1.
3. Elimination ($N-1$, L_N-1).
4. Remove G from L_N.

We also applied the automatic loop-breaker selection software developed by Becker et al. to the pedigree in figure 2, to determine how the number of consistent loop-breaker vectors would change with this choice of loop breakers (marked by solid gray symbols). It is interesting to note that, although this choice of loop breakers still has the *same* number of loop-breaker vectors (using set-recoded genotypes) as the original pedigree after the initial genotype elimination (49,152), the number of consistent loop-breaker vectors is reduced from 14,270 to 2,240. This more-than-sixfold reduction is due to the fact that these loop breakers are more closely related than the others (three of them are siblings, a fourth one is their father, and a fifth one is a daughter), so that the ordered genotypes in the recursion reduce the genotype lists of the remaining loop breakers even further than before.

Detecting Genotyping Errors

In practical applications, genotype elimination can also be useful for detection of genotype errors prior to likelihood computation (with programs such as PedCheck [O'Connell and Weeks 1998]). Detection of someone *typed* with a superfluous genotype is, however, a different issue from elimination of superfluous genotypes from the genotype list of an untyped individual; the first has to do with efficient detection of errors, whereas the second has to do with efficient likelihood computations. Thus we were interested in exploring whether the conventional Lange-Goradia genotype-elimination algorithm would suffice to detect anyone typed with only a superfluous genotype on a pedigree with loops and thereby identify that the pedigree had a Mendelian inconsistency. If so, then we could conclude that the pedigree is inconsistent, without having to deal with the multiplicative increase in computing required when dealing with loop-breaker vectors. For our pedigree in figure 1, with the single loop formed by the sibling mating, if we type either grandparent with either superfluous genotype 1/1 or 3/3, the conventional Lange-Goradia algorithm will detect the error. Unfortunately, we have found a counterexample, displayed in figure 3, with a single loop arising from a first-cousin mating where conventional Lange-Goradia genotype-elimination fails to

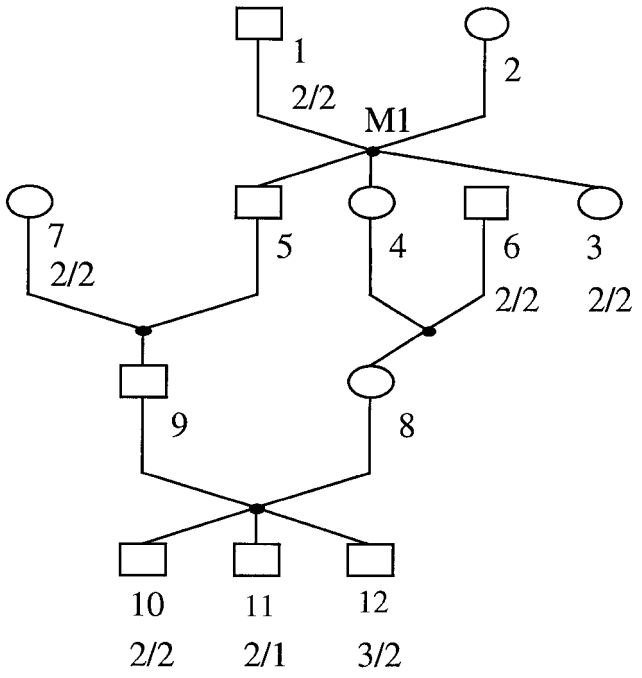

Figure 3 Pedigree with one loop that illustrates that Lange-Goradia genotype elimination alone is not able to detect that the pedigree is not consistent with the laws of Mendelian inheritance.

detect that each of the seven typed individuals has a superfluous genotype (table 2). For example, the genotype 2/2 in individual 1 is superfluous, because the genotype forces individual 2 to have three distinct alleles. This inconsistency arises because the known genotypes of the rest of the pedigree imply the following allele flow through the mating vertex M1 to individual 2: alleles 1 and 3 from individuals 4 and 5 combined and allele 2 from individual 3. Thus, determining whether a pedigree is consistent with the laws of Mendelian inheritance requires the use of our new algorithm. In addition to determining whether a pedigree is consistent with the laws of Mendelian inheritance, PedCheck also uses our new algorithm to help identify possible sources of genotype error, by determining which genotyped individuals would eliminate the inconsistency if their genotype information were actually missing (called "critical genotypes") and then computing a likelihood-based odds for each possible alternative genotype that the individual could have. In figure 3, PedCheck determined that, of the seven typed individuals, only individual 10 does not have a critical genotype and that either individual 11 or 12 is the most likely source of the error. The reason is that, in the original genotype data, seven of the eight founder alleles are 2, and, for this pedigree, the likelihood (when equifrequent alleles are used) is greater when there is only one other allele besides the 2 (the case when individual 11 or 12 has the critical genotype) than when there are two other distinct alleles (as is the case when individual 1, 3, 6, or 7 has the critical genotype).

Discussion

We have presented here a genotype-elimination algorithm that is guaranteed to eliminate all superfluous genotypes in all types of pedigrees, including those with loops. Although this should help in accelerating the computations of likelihoods in pedigrees with loops, computations in pedigrees with more than a few loops will continue to remain challenging.

We have discussed the computational complexity of the Lange-Goradia algorithm and have presented a more efficient implementation to improve the speed of this algorithm, which is repeated many times in our optimal genotype-elimination algorithm. We also showed that set recoding can reduce the number of loop-breaker vectors when there are untyped loop breakers. Also, we have illustrated, in our pedigree with six loops, that many of the loop-breaker vectors were inconsistent and have presented a recursive genotype-elimination algorithm to eliminate the need to check these vectors. We also have shown that the choice of loop breakers could have a dramatic effect on the number of consistent loop-breaker vectors, and we plan to further investigate algorithms for choosing an optimal loop-breaker set in the context of set recoding and our recursive loop breaker–elimination algorithm.

Finally, we have presented an application of genotype elimination to detection genotyping errors in pedigrees and have shown, with a counterexample, that the original Lange-Goradia algorithm can not detect every inconsistency. However, our new genotype-elimination algorithm is guaranteed to detect every Mendelian in-

Table 2

Ordered Genotype Lists after Lange-Goradia Genotype Elimination Is Applied to the Pedigree in Figure 3

Person	Genotype List
1	{2|2}
2	{2|1, 1|2, 3|2, 2|3}
3	{2|2}
4	{2|1, 2|3}
5	{2|1, 2|3}
6	{2|2}
7	{2|2}
8	{2|1, 2|3}
9	{1|2, 3|2}
10	{2|2}
11	{2|1, 1|2}
12	{3|2, 2|3}

consistency efficiently and quickly. We have shown how its implementation in PedCheck can assist the researcher in identification of the possible sources of genotype errors.

Software

Our new algorithms have been implemented into the software packages PedCheck and VITESSE, which can be obtained from our Division of Statistical Genetics website.

Acknowledgments

Pedigree figures were generated with Pedigree/Draw version 4.4 (Southwest Foundation for Biomedical Research). We thank Toby Nygaard for permission to use the pedigree from Saudi Arabia. This work was supported, in part, by funds from NIH grant HG00932, National Institute of Aging grant AG16992-01, BIOMED European Community grant PL 96 2532, the Wellcome Trust Centre for Human Genetics at the University of Oxford, the University of Pittsburgh, the W. M. Keck Center for Advanced Training in Computational Biology at the University of Pittsburgh, Carnegie Mellon University, and the Pittsburgh Supercomputing Center.

Electronic-Database Information

The URL for data in this article is as follows:

Division of Statistical Genetics, Department of Human Genetics, University of Pittsburgh, http://watson.hgen.pitt.edu (for PedCheck and VITESSE software)

References

Becker A, Geiger D, Schaffer AA (1998) Automatic selection of loop breakers for genetic linkage analysis. Hum Hered 48:49–60

Cottingham RW Jr, Idury RM, Schäffer AA (1993) Faster sequential genetic linkage computations. Am J Hum Genet 53: 252–263

Jones AC, Yamamura Y, Almasy L, Bohlega S, Elibol B, Hubble J, Kuzuhara S, et al (1998) Autosomal recessive juvenile parkinsonism maps to 6q25.2-q27 in four ethnic groups: detailed genetic mapping of the linked region. Am J Hum Genet 63:80–87

Lange K, Boehnke M (1983) Extensions to pedigree analysis. V. Optimal calculation of Mendelian likelihoods. Hum Hered 33:291–301

Lange K, Elston RC (1975) Extensions to pedigree analysis. I. Likelihood calculations for simple and complex pedigrees. Hum Hered 25:95–105

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

Lange K, Weeks DE (1989) Efficient computation of LOD scores: genotype elimination, genotype redefinition, and hybrid maximum likelihood algorithms. Ann Hum Genet 53: 67–83

Lathrop GM, Lalouel JM (1988) Efficient computations in multilocus linkage analysis. Am J Hum Genet 42:498–505

O'Connell JR, Weeks DE (1995) The VITESSE algorithm for rapid exact multilocus linkage analysis via genotype set-recoding and fuzzy inheritance. Nat Genet 11:402–408

——— (1998) PedCheck: a program for identification of genotype incompatibilities in linkage analysis. Am J Hum Genet 63:259–266

Schäffer AA (1996) Faster linkage analysis computations for pedigrees with loops or unused alleles. Hum Hered 46: 226–35

Sobel E, Lange K, O'Connell JR, Weeks DE (1995) Haplotyping algorithms. In: Speed TP, Waterman MS (eds) Genetic mapping and DNA sequencing: IMA volumes in mathematics and its applications. Springer-Verlag, New York

Xie X, Ott J (1992) Finding all loops in a pedigree. Am J Hum Genet Suppl 51:A206