Human Heredity

Hum Hered 2000;50:43–56

Received: February 17, 1999
Revision received: April 26, 1999
Accepted: May 3, 1999

Haplotyping in Pedigrees via a Genetic Algorithm

Pradip Tapadar Saurabh Ghosh Partha P. Majumder

Indian Statistical Institute, Calcutta, India

Key Words
Multilocus haplotype · Recombinant · Optimization · Pedigree analysis

Abstract
Genome-wide screening for localization of disease genes necessitates the efficient reconstruction of haplotypes of members of a pedigree from genotype data at multiple loci. We propose a genetic algorithmic approach to haplotyping and show that it works fast, efficiently and reliably. This algorithm uses certain principles of biological evolution to find optimal solutions to complex problems. The optimality criterion used in the present problem is the minimum number of recombinations over possible haplotype configurations of members of a pedigree. The proposed algorithm is much less demanding in terms of data and assumption requirements compared to the currently used likelihood-based methods of haplotype reconstruction. It also provides multiple optimal haplotype configurations of a pedigree, if such multiple optima exist.

Introduction

Because of the tremendous successes in localization of disease genes using multipoint mapping techniques, genome-wide screening has become very popular. Genome-wide screening studies yield data on sets of several linked polymorphic marker loci in members of families. These data usually necessitate the reconstruction of haplotypes for identifying, among others, the smallest genomic region containing the disease gene and also genotyping errors. Several systematic methods for haplotype reconstruction have already been suggested. Some of these are rule-based [1, 2], while others are based on likelihood computations [3–7]. While there are some advantages of using likelihood-based methods over rule-based methods [8], rule-based methods are, however, usually less demanding in terms of data and assumptions, and can be computationally much faster than likelihood-based methods, especially for large pedigrees.

In this study, we propose a rule-based method for haplotype reconstruction in pedigrees using a genetic algorithmic approach for optimization. We also provide results on the efficiency of the proposed method: The only data that are required in the proposed method are (1) pedigree structure, and (2) genotypes of family members at the various loci. Estimates of allele frequencies and map/recombination distances and the assumptions of

Hardy-Weinberg equilibrium and linkage equilibrium that are generally required by likelihood-based methods of haplotyping are not required by the proposed method.

Multilocus Haplotyping

Consider a set of linked loci. Given data on genotypes of members of a family, the problem of haplotyping is to assign alleles to each of the two chromosomes of every individual in the family in some optimal way. For example, if A_1A_2, B_1B_2, C_1C_3 denote the genotypes of an individual at three linked loci, then the possible haplotypes of this individual are: (1) $A_1B_1C_1/A_2B_2C_3$, (2) $A_1B_1C_3/A_2B_2C_1$, (3) $A_1B_2C_1/A_2B_1C_3$, and (4) $A_1B_2C_3/A_2B_1C_1$. (We have used the conventional notation (1) $A_1B_1C_1/A_2B_2C_3$ to denote that the alleles, A_1, B_1 and C_1 are on one chromosome and the alleles A_2, B_2 and C_3 are on the homologous chromosome.) Now, suppose the haplotypes of the parents of this individual are $A_1B_1C_1/A_2B_2C_2$ and $A_1B_1C_1/A_2B_2C_3$. Then, for the possible haplotype assignments for this individual, the total minimum numbers of recombinations (R) are: (1) R = 0, (2) R = 2, (3) R = 4 and (4) R = 2. One way of determining haplotypes optimally is to choose that combination of haplotypes of members for which the total number of recombination events in the entire family is the minimum. Hence, the optimal haplotype assignment is (1); haplotype assignments (2) and (4) are equally suboptimal and haplotype assignment (3) is the 'worst'. Obviously, as exemplified above, optimal determination of haplotypes involves a complicated search procedure. If the numbers of loci or/and family members are large, an exhaustive search of the set of all possible haplotypes of the family members for determining the optimal haplotype configuration of the family is practically impossible. We propose a genetic algorithm (GA) approach to determine the optimal haplotypic configuration of a family which, as we shall show, is computationally extremely efficient.

GAs: Philosophy and Basic Principles

GAs [9, 10] are robust and efficient search and optimization techniques that are motivated by evolutionary principles. In Darwinian evolution, starting with a large number of individuals with differing genotypes and, therefore, differing fitnesses, new generations are formed by reproduction. Relative frequencies of fitter genotypes increase in every generation. The average fitness of the population, therefore, increases over generations. The biological system thus converges to an optimum, in which all members of the population are of the same maximally fit genotype. (We, however, note that there are situations when these general evolutionary properties may be violated. With linked epistatically interacting loci, for example, mean fitness may not increase.) Mutation and crossover introduce genetic variability in the population over generations, ensuring thereby that the system does not get stuck at a local optimum.

The basic steps in the actual implementation of GAs are described below. Suppose a function $f(x)$ is to be optimized (say, maximized) over a closed bounded set A. For simplicity of exposition, let us assume $A = [a, b]$ on the real line. Then for any level of precision of maximization $\epsilon > 0$, there exists an integer N such that $h = \frac{b-a}{2^N-1} < \epsilon$. Let $m = 2^N$. The search space $[a, b]$ can now be modified to a search space comprising a finite number of points $a = a_0 < a_1 < ... < a_m = b$, by dividing the interval $[a, b]$ into subintervals $[a_0, a_1], [a_1, a_2], ..., [a_{m-1}, a_m]$, where $a_i - a_{i-1} = h < \epsilon (i = 1,2,...,m)$. If ϵ is chosen to be sufficiently small, then optimizing $f(x)$ over the discretized search space ensures a small margin of error.

The modified search space comprises the population; $a_0, a_1, ..., a_m$ are individuals of the population. The individuals are represented by binary strings; a_i is represented by the binary representation of i, appropriately left-adjusted by zeroes so that the string is of length N. Having initialized the population, GAs comprise the following steps until convergence: (a) generate a random sample of $2n$ individuals (strings) and obtain their fitness values [values of $f(x)$]. (b) Perform 'reproduction' which is a 'genetic operator' in which $2n$ strings are first selected from the initial population with probabilities proportional to their fitness values. These strings are then paired at random (mating partners) and entered into a mating pool. (c) To individuals in the mating pool, two 'genetic operations' – crossover and mutation – are performed with probabilities p_c and p_m, respectively. For implementation of the 'crossover' operator, a string is first selected with probability p_c. Then, an integer position k along the string is selected uniformly at random between 1 and $N - 1$. Two new strings are created by swapping (i.e., interchanging) all characters between positions $k + 1$ and N inclusively. The 'mutation' operator is implemented by changing the value of a character in a particular position of a string with a probability p_m. In the binary coding, this simply means changing a 1 to a 0 and vice versa.

Both operators, crossover and mutation, introduce and sustain diversity in the population. The choices of p_c and

p_m are known to critically affect the behavior and performance of the GA [10, 11]. Typical values of p_c are in the range 0.5–1.0. However, p_m is typically chosen in the range 0.005–0.05 because large values of p_m transform the GA to a purely random search algorithm. For multimodal functions, such as the number of recombination events in a family, it is recommended [12] that instead of holding p_c and p_m at fixed values, these probabilities be varied adaptively in response to the fitness values of the solutions. We have used this recommendation as will be explained subsequently.

A Genetic Algorithm for Haplotyping

In this section, we shall provide details of the proposed GA for the haplotyping problem. The data given are: (1) the family structure, and (2) genotypes of every member of the family at each of L loci. In the GA, an 'individual' will be a family of the same structure as given. As stated earlier, the fitness function $f(.)$ will be defined on a family as the number of recombination events required to explain a specific haplotypic configuration of the members of this family inferred on the basis of the genotypic data given. Our objective is to obtain a haplotypic configuration for which $f(.)$ is the minimum.

The Binary Representation and Some Basic Strategies

To apply GA, we have to define a binary string for every member of the family. But first we provide a possible scheme for coding the positions of crossover. Consider a nuclear family comprising two parents and an offspring. Suppose the parents are artitrarily numbered as 1 and 2, and the offspring as 3. For ease of exposition, we label two alleles at locus l of each member of the family distinctly: A_lB_l for parent 1, C_lD_l for parent 2 and E_lF_l for the offspring. We construct haplotypes of each parent by randomly assigning an allele to a chromosome for every locus l ($l = 1,2,...,L$), consistent with her/his observed genotypes. Without loss of generality, we may assume that the haplotype of parent 1 is $A_1A_2...A_L/B_1B_2...B_L$ and that for parent 2 is $C_1C_2...C_L/D_1D_2...D_L$. (We, however, emphasize that knowledge of parental haplotypes is assumed at this stage only for ease of exposition of the algorithm. In actual implementation of the algorithm, all possible parental haplotypes will be considered and evaluated.) Given the haplotypes of the parents, we wish to determine the haplotypes of the offspring in some optimal way. Let us consider the chromosomes of an individual as an ordered pair (O_1, O_2), where O_1 is the chromosome transmitted by parent 1 and O_2 is the chromosome transmitted by parent 2. We shall call O_1 the 1st chromosome and O_2 the 2nd. If the individual is a founder, the parental origin of chromosomes remains indeterminate.

Define $s_i(l)$ = identity of the chromosome of the ith member used to haplotype the lth locus of the offspring; $i = 1,2$; $l = 1,2,...,L$. (We shall use the phrase 'haplotype the lth locus' to mean 'assign alleles of the lth locus to specific chromosomes'.) Based on different genotypic possibilities, we discuss a sequential procedure to haplotype each of the L loci of the offspring, and in each case, prescribe an optimal strategy in terms of $s_i(l)$. $s_i(l)$ can take values 0, 1 or 2. Rules for assignment of these values are discussed in the paragraphs that follow.

Initially, prior to any assignment of chromosomes, we set $s_i(0) = 0$; $i = 1, 2$. We present in table 1 the strategy to haplotype the lth locus of the offspring separately for the cases $s_i(l-1) = 0$ and $s_i(l-1) \neq 0$. Note that the genotypes of the two parents at the lth locus are A_lB_l and C_lD_l repectively, while that of the offspring is E_lF_l.

We need a separate treatment for the case when $A_lB_l = C_lD_l = E_lF_l$ and $A_l \neq B_l$, $C_l \neq D_l$, $E_l \neq F_l$. In this case, we first examine whether the strategies $s_1(l) = s_1(l-1)$ are consistent with Mendelian compatibility. If consistent, we use this strategy. If not, we randomly allocate E_l and F_l to chromosomes 1 and 2 of the offspring. If E_l is assigned to chromosome 1, then $s_1(l) = 1$ or 2 according as $A_l = E_l$ or $B_l = E_l$ and $s_2(l) = 1$ or 2 according as $C_l = F_l$ or $D_l = F_l$. If F_l is assigned to chromosome 1, then $s_1(l) = 1$ or 2 according as $A_l = F_l$ or $B_l = F_l$ and $s_2(l) = 1$ or 2 according as $C_l = E_l$ or $D_l = E_l$. However, we note that one random assignment of the alleles to the two chromosomes may lead to an increase in the number of recombinants at a subsequent stage, when it might be avoided for another random assignment. Thus if for a subsequent locus, assignment of alleles to chromosomes at that locus can be done without randomization, we retrace back those loci at which alleles were randomly assigned and examine whether any other random assignment can reduce the number of recombinants.

We note that unambiguous assignment of alleles to chromosomes may not be possible at every locus. Therefore, starting with the first locus, we refrain from actually assigning alleles to chromosomes until we reach a locus at which unambiguous assignment is possible. When this unambiguous assignment is completed, then we fix that chromosome for all previous loci. If $s_i(L) = 0$, that is both chromosomes are 'feasible' for all loci, we set, without loss of generality, $s_i(l) = 1$, $\forall l = 1,2,...,L$; $i = 1,2$. Moreover $s_i(l) = s_i(l-1)$ indicates that while assigning alleles to

Table 1. Strategies for assigning alleles at the lth locus of an offspring to chromosomes when alleles at the $(l-1)$th locus have already been assigned

Condition on genotypes	Strategy when $s_1(l-1)=0$	Strategy when $s_1(l-1)\neq 0$	Strategy when $s_2(l-1)=0$	Strategy when $s_1(l-1)\neq 0$
$A_l = B_l$, $C_l = D_l$	$s_1(l) = s_1(l-1)$	$s_1(l) = s_1(l-1)$	$s_2(l) = s_2(l-1)$	$s_2(l) = s_2(l-1)$
$A_l = B_l$, $C_l \neq D_l$, $E_l/F_l = C_l$	$s_1(l) = s_1(l-1)$	$s_1(l) = s_1(l-1)$	$s_2(j) = 1;\ \forall_j \leq l$	$s_2(l) = 1$
$A_l = B_l C_l \neq D_l$, $E_l/F_l = D_l$	$s_1(l) = s_1(l-1)$	$s_1(l) = s_1(l-1)$	$s_2(j) = 2;\ \forall_j \leq l$	$s_2(l) = 2$
$A_l \neq B_l$, $C_l = D_l$, $E_l/F_l = A_l$	$s_1(l) = 1;\ \forall_j \leq l$	$s_1(j) = 1$	$s_2(l) = s_2(l-1)$	$s_2(l) = s_2(l-1)$
$A_l \neq B_l$, $C_l = D_l$, $E_l/F_l = B_l$	$s_1(j) = 2;\ \forall_j \leq l$	$s_1(l) = 2$	$s_2(l) = s_2(l-1)$	$s_2(l) = s_2(l-1)$
$A_l \neq B_l$, $C_l \neq D_l$, $(A_lB_l = C_lD_l = E_lF_l)^c$ $E_l/F_l = A_l$, $E_l/F_l = C_l$	$s_1(j) = 1;\ \forall_j \leq l$	$s_1(l) = 1$	$s_2(j) = 1;\ \forall_j \leq l$	$s_2(l) = 1$
$A_l \neq B_l$, C_l, $(A_lB_l = C_lD_l = E_lF_l)^c$ $E_l/F_l = A_l$, $E_l/F_l = D_l$	$s_1(j) = 1;\ \forall_j \leq l$	$s_1(l) = 1$	$s_2(j) = 2;\ \forall_j \leq l$	$s_2(l) = 2$
$A_l \neq B_l$, $C_l \neq D_l$, $(A_lB_l = C_lD_l = E_lF_l)^c$ $E_l/F_l = B_l$, $E_l/F_l = C_l$	$s_1(j) = 2;\ \forall_j \leq l$	$s_1(l) = 2$	$s_2(j) = 1;\ \forall_j \leq l$	$s_2(l) = 1$
$A_l \neq B_l$, $C_l \neq D_l$, $(A_lB_l = C_lD_l = E_lF_l)^c$ $E_l/F_l = B_l$, $E_l/F_l = D_l$	$s_1(j) = 2;\ \forall_j \leq l$	$s_1(l) = 2$	$s_2(j) = 2;\ \forall_j \leq l$	$s_2(l) = 2$

At the lth locus (E_l, F_l) denote the genotypes of the offspring with parental genotypes (A_l, B_l) and (C_l, D_l).

chromosomes at the lth locus, we assign the same parental chromosome as for the $(l-1)$th locus, that is, there is no crossover.

Following the above procedure, we determine haplotypes for the nonfounders given the haplotypes of their parents. Suppose we number the members in the pedigree as $1,2,...,K$. Let the set of nonfounders in the pedigree be denoted by $\bar{F}$. Clearly $\bar{F} \subset \{1,2,...,K\}$. Consider an individual $i \in \bar{F}$. Let the parents of individual i be I_1 and I_2 where $I_1 < I_2$. For each $i \in \bar{F}$, we now define $2(L-1)$ binary variables X_{ij}, $j = 1,2,...,2(L-1)$.

For $j = 1,2,...,L-1$, X_{ij} is defined with respect to I_1 and for $j = L, L+1,...,2(L-1)$, X_{ij} is defined with respect to I_2. We assign values of X_{ij} as follows:

$$X_{ij} = 0, \text{ if } s_{I_1}(j+1) = s_{I_1}(j), j \leq L-1 \text{ or } s_{I_2}(j+2-L) = s_{I_2}(j+1-L), j \geq L$$
$$= 1, \text{ otherwise.}$$

Let $X_i = (X_{i1}, X_{i2},..., X_{i,2(L-1)})$; $i \in \bar{F}$. The objective function of our problem can be expressed in terms of X_{ij}. Let $\mathbf{X}$ be a vector of 0s and 1s given by $(X_1, X_2,...)$. Then the objective function is given by:

$$f(\mathbf{X}) = \sum_{i \in \bar{F}} \sum_{j=1}^{2(L-1)} X_{ij}$$
$$= \text{number of recombinants in the pedigree.}$$

Thus using the binary strings $X_1, X_2,...$, we implement the different genetic operators of GA to arrive at an optimal solution to our problem.

Some examples: Before proceeding further, we provide some simple examples to clarify concepts and notations. Consider a three-member nuclear family with one offspring. Suppose each individual is genotyped at three autosomal codominant loci. Let the genotypes of the father (ID 1) be: 11, 11 and 12 at the three loci, respectively. Let the genotypes of the mother (ID 2) be 22, 12 and 34; and those of the offspring (ID 3) be 12, 12 and 13 (fig. 1a). Since we do not know the phase of the parents, we begin by randomly assigning their alleles to their chromosomes. For such an assignment, we attempt to determine haplotypes of the offspring. (This procedure is repeated with all other possible assignments of alleles of parents to their chromosomes in order to arrive at optimal haplotype configurations.) Suppose the random assignment of alleles to chromosomes yields the haplotype 111/112 for the father and 213/224 for the mother (fig. 1a). We now wish to determine haplotypes of the offspring.

At locus 1, the offspring is of genotype 12. While it is clear that at this locus the offspring has received the allele 1 from the father and the allele 2 from the mother, at this stage the identities of the paternal and maternal chromo-

somes received by the offspring remain unclear. That is, the offspring may have received either the 1st or the 2nd chromosome from the father. (We recall the convention that we have followed: if an individual's haplotypes are written as 111/112, then we label the '111'-chromosome as the 1st and the '112'-chromosome as the 2nd.) Hence, at locus 1 of the offspring, the identity of the chromosome derived from the father (ID 1) is inderminate (that is, either chromosome of the father is 'feasible'); therefore, $s_1(1) = 0$. Similarly, $s_2(1) = 0$, since both chromosomes of the mother are 'feasible' at locus 1 of the offspring.

At locus 2, the offspring is of genotype 12. It is clear that allele 1 is of paternal origin and allele 2 is of maternal origin. It is also clear now that the offspring has received allele 2 from the 2nd chromosome of the mother. We thus set $s_2(2) = 2$ and reset $s_2(1) = 2$. Since the identitiy of the maternal chromosome was indeterminate at the previous state (locus 1), we had set $s_2(1) = 0$. But because we have now identified that the offspring has received the allele at locus 2 from the 2nd chromosome (that is, the '224'-chromosome) of the mother, we reset $s_1(1)$ from 0 to 2, since this is the most parsimonious solution at this stage. Because even at this stage, the identity of the chromosome transmitted by the father remains ideterminate, we set $s_1(2) = 0$ [$= s_1(1)$].

The genotype of the offspring is 13 at locus 3. Clearly, allele 1 is paternally derived and allele 3 is maternally derived. Allele 1 is on the 1st chromosome of the father; hence, we set $s_1(3) = 1$ and, for reasons stated in the earlier paragraph, we reset $s_1(1) = s_1(2) = 1$. Allele 3 is derived from the 1st chromosome of the mother; hence, we set $s_1(3) = 1$. (No further resetting is done as all $s_i(l)$ values are non-zero at this stage. At any stage, only such $s_i(l)$s are considered eligible for resetting that have values equal to 0, i.e., 'indeterminate'.)

Now, as $s_1(1) = s_1(2)$, $X_{31} = 0$; as $s_1(2) = s_1(3)$, $X_{32} = 0$; as $s_2(1) = s_2(2)$, $X_{33} = 0$; and as $s_2(2) \neq s_2(3)$, $X_{34} = 1$. Then, $f(\mathbf{X}) = X_{31} + X_{32} + X_{33} + X_{34}$ = total number of recombinants = 1 (fig. 1a).

To explain how we deal with a possible complication that arises when both parents and the offspring are heterozygous for the same alleles at a locus, we consider another small example of a similar three-member nuclear family. Suppose the randomly assigned haplotypes, at two codominant biallelic loci, are 11/22 for the father and 13/24 for the mother. Suppose the genotypes of the offspring are 12 and 14 (fig. 1b).

At locus 1, the offspring and parents are all heterozygotes 12. Hence, determining which allele in the offspring is derived from which paternal chromosome is clearly not

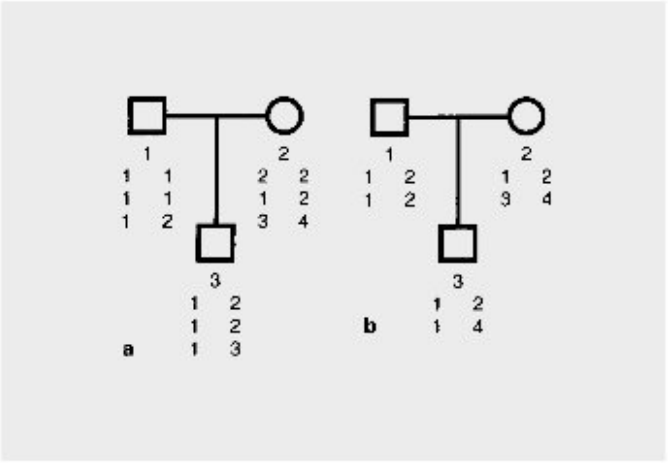


Fig. 1. Nuclear family used for exemplifying the basic strategies (**a**), and a possible complication of the proposed genetic algorithm for haplotyping (**b**). Reconstructed haplotypes are depicted under each individual.

possible. We consider two possible scenarios to indicate how we have handled such situations.

Case I: $s_1(1) = 2$ and $s_2(1) = 1$; that is, at locus 1, allele 2 is transmitted via the 2nd chromosome of the father and allele 1 is transmitted via the 1st chromosome of the mother. Now, at locus 2, the offspring's genotype is 14. At this locus, clearly allele 1 is paternally derived and allele 4 is maternally derived. Hence, $s_1(2) = 1$ and $s_2(2) = 2$. Therefore, $X_{31} = 1$ as $s_1(1) \neq s_1(2)$ and $X_{32} = 1$ as $s_2(1) \neq s_2(2)$ and $f(\mathbf{X}) = X_{31} + X_{32} = 2$.

Case II: $s_1(1) = 1$ and $s_2(1) = 2$; that is, at locus 1, allele 1 is transmitted via the 1st chromosome of the father and allele 2 via the 2nd chromosome of the mother. At locus 2, $s_1(2) = 1$ and $s_2(2) = 2$. Hence, as $s_1(1) = s_1(2)$, $X_{31} = 0$ and as $s_2(1) = s_2(2)$, $X_{32} = 0$; thus, $f(\mathbf{X}) = 0$. As case II yields a lower value of $f(\mathbf{X})$ than case I, we accept this scenario as the more parsimonious one.

Thus, in ambiguous situations, we exhaustively consider all possible cases in our algorithm to arrive at the most parsimonious solution. If multiple cases result in the same value of $f(\mathbf{X})$, we choose one of them randomly with equal probability. The arguments used for haplotyping pedigrees are straightforward extensions of those exemplified above for nuclear families. To each founder in a pedigree, we assign all possible phases and determine haplotypes of offspring as exemplified above using strategies provided in table 1. Minimum total number of recombination events in the entire pedigree is used as the optimality criterion.

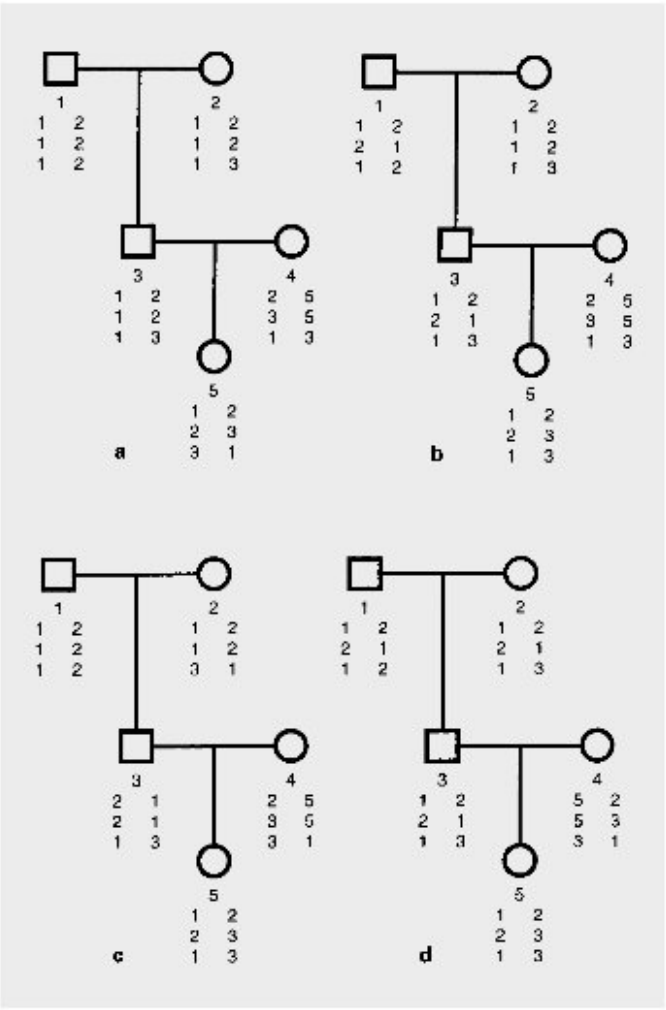


Fig. 2a–d. Pedigrees used for exemplifying the methods and applications of the reproduction and crossover operators.

To arrive at an optimal solution in a computationally efficient manner, that is, without exhaustive enumeration, while simultaneously ensuring that only a locally optimum solution is not declared as the final solution, we introduce several operators.

Reproduction Operator

We start with a number of binary strings representing the candidates in the first GA generation. To obtain an initial string, we randomly assign phases to founders of the family. Then for any other member, we assign the haplotype by considering his/her parents as described in the previous section. In this manner, we generate a possible haplotypic configuration for each member in the family and obtain the binary string containing the information about recombinations. In the first generation, we simulate $2M$ such candidate haplotypes, where $M > 0$ is a sufficiently large integer.

Next, we select the fitter strings and introduce them in the mating pool where crossover and mutation operators can act to produce the second generation of candidates. The aim is to design a scheme to choose fitter strings. In our definition, fitter candidates are those with lesser number of recombinants. Therefore, a candidate with a fewer number of recombinants is chosen with a higher probability.

We propose the following scheme to choose fitter candidates:

Define Y_i = number of recombinants for ith candidate string, $i = 1,2,...,2M$.

Let:

$$Z_i = 1, \quad \text{if } Y_i = max_j Y_j,$$
$$1 - (Y_i - max_j Y_j), \quad \text{if } Y_i \neq max_j Y_j$$

The fitness function is defined as:

$$F(Y_i) = \frac{Z_i}{\Sigma_{j=1}^{2M} Z_j}$$

In this scheme, candidates with a low number of recombinants have high fitness values and those with a high number of recombinants have low fitness values. Thus, maximizing $F(.)$ is equivalent to minimizing $f(.)$. (We note that an alternative and a more intuitive scheme would be to define the fitness function to be:

$$F_1(Y_i) = \frac{1/Y_i}{\Sigma_{j=1}^{2M} 1/Y_j}.$$

However for high values of Y_i, this fitness function does not exhibit a substantial increase with decrease in Y_i which is reflected in our proposed scheme.) Under the proposed scheme, the ith candidate string is selected with probability $F(Y_i)$. Thus we randomly select, with replacement, $2M$ candidates as the tentative new population and introduce them in the mating pool.

An example: We now provide an example with a 5-member pedigree as shown in figure 2. We emphasize that only genotypes of each member are provided as input data. Thus, the genotypes of ID 1 at the three loci are 12, 12 and 12; those of ID 2 are 12, 12 and 13, etc. To keep the example simple, we choose $M = 2$; that is, four families (fig. 2a–d). (We emphasize that we have chosen a small value of M only to keep the example simple. Our recommendation, discussed in later sections, in actual practice

is that M should equal at least the total number of individuals in the pedigree.) In each family, in the first GA generation, we randomly assign phase to founders, and determine haplotypes of other members using the principles and procedures outlined earlier. Figures 2a–d provide these results for 4 candidates. The values of Y_i = number of recombinants are, for the candidate families drawn in figures 2a–d, $Y_1 = 1$, $Y_2 = 3$, $Y_3 = 2$ and $Y_4 = 1$. Therefore, $max_j\ Y_j = Y_2 = 3$. Hence, $Z_1 = 1 - (Y_1 - Y_2) = 3$, $Z_2 = 1$, $Z_3 = 1 - (Y_3 - Y_2) = 2$, $Z_4 = 1 - (Y_4 - Y_2) = 3$. $Z_1 + Z_2 + Z_3 + Z_4 = 9$. Thus, the fitnesses of the candidates are: $F(Y_1) = 1/3$, $F(Y_2) = 1/9$, $F(Y_3) = 2/9$ and $F(Y_4) = 1/3$. $F(Y_1) + F(Y_2) + F(Y_3) + F(Y_4) = 1$. It may be noted that Z_i and $F(Y_i)$ increase as Y_i decreases. Thus, the fitness of a candidate family is higher if this family has a smaller number of recombinants.

To form the new mating pool, we draw four random numbers from Uniform[0,1] distribution. If a number drawn lies between 0 and 1/3, we choose family 1; if between 1/3 and 4/9, we choose family 2; if between 4/9 and 2/3, we choose family 3; and if between 2/3 and 1, we choose family 4. Suppose, for the purpose of exemplification, the families chosen are 1, 4, 3 and 4. (Obviously, candidates with higher fitnesses will be overrepresented in the mating pool.) To form the new GA generation, we consider two further operators.

Crossover Operator

A direct application of the crossover operator used in conventional GA does not work for the present problem. In conventional GA, the crossover operator, introduced to enhance genetic diversity in the population, comprises choosing two binary strings and swapping portions of the strings. For the present problem, such a swapping operation is meaningless for enhancing genetic diversity in the population and may also result in Mendelian incompatibility. We thus need to suitably modify the conventional crossover operator.

We choose a pair of candidate strings and apply the crossover operator using a specified probability mechanism. We assign a probability p_c to the occurrence of a crossover, that is, with probability $(1 - p_c)$ the crossover operation is not performed and the candidate strings are passed on to the next generation. For performing the crossover operation (with probability p_c), we choose a founder randomly from the given pedigree and find its mating partner. If there is no mating partner, both the candidate strings are passed on to the next generation. If the mating partner is also a founder, then this crossover is similar to the conventional GA crossover. We interchange the haplotypes of these members and make necessary adjustments for their descendents to maintain Mendelian compatibility. If the mating partner is not a founder, we perform the crossover operation using the following scheme. We observe the number of offspring (N_0) of these two members. We interchange the haplotypes of these mating partners with probability $N_0/(N_0 + 2)$ and make the necessary adjustments (as exemplified below) for the parents of the nonfounder mating partner. Similarly with probability $2/(N_0 + 2)$, we interchange the haplotypes of the founder only and thus need to make adjustments for only the descendents.

Table 2. Adjustments prescribed after performing 'crossover' operation

Case	Prescription
$A_f = B_f = E_f$	$s_1(l) = s_1(l-1)$
$A_f = E_f$, $B_f \neq E_f$	$s_1(l) = 1$
$B_f = E_f$, $A_f \neq E_f$	$s_1(l) = 2$
$C_f = D_f = F_f$	$s_2(l) = s_2(l-1)$
$C_f = F_f$, $D_f \neq F_f$	$s_2(l) = 1$
$D_f = F_f$, $C_f \neq F_f$	$s_2(l) = 2$

Recall the set-up of a nuclear family of 3 individuals described in an earlier section ('Haplotyping'). Note that while performing the crossover operation, the haplotype of member 3 is already fixed, that is, we know that $E_1E_2...E_L$ and $F_1F_2...F_L$ are transmitted from member 1 and member 2, respectively. Let $s_i(l)$ be as defined earlier. We prescribe the required adjustments in terms of $s_i(l)$ in table 2. If $s_i(l) = s_i(l-1)$, we assign 0 to the respective element in the binary string and if $s_i(l) = 3 - s_i(l-1)$, we assign 1 to that element in the string.

While implementing the crossover operator, we use adaptive p_c values as:

$$p_c = \frac{f' - f_{min}}{\bar{f} - f_{min}}, \text{ if } f' \leq \bar{f}$$
$$= 1, \text{ otherwise;}$$

where f' = minimum of the number of recombinants of the two candidates (nuclear families) between which the crossover operation is to be performed, $\bar{f}$ = average number of recombinants in the previous generation, and f_{min} = the minimum number of recombinants attained in the previous generation. Note that p_c is a bivariate function of the numbers of recombinants in the two candidates (nuclear families) between which the crossover operation is to be performed.

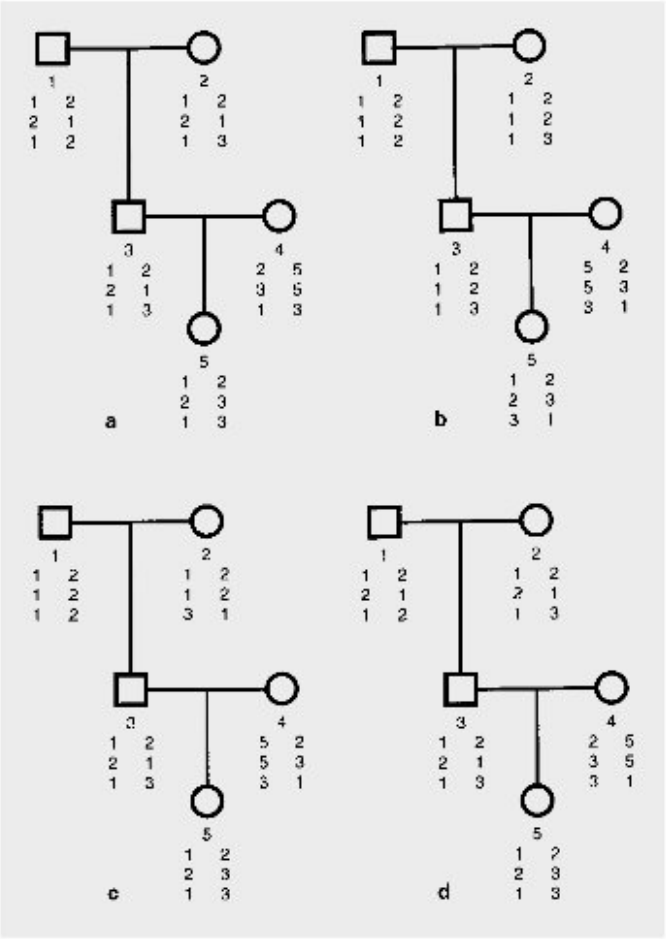


Fig. 3a–d. Post-crossover example pedigrees (fig. 2a–d) which are also used for exemplifying the method and application of the mutation operator.

An illustration: To illustrate this operator, we continue with the example described in the previous section. The mating pool comprised families 1 (fig. 2a), 4 (fig. 2d), 3 (fig. 2c) and 4 (fig. 2d). Before forming the new GA generation, we introduce 'genetic diversity' through the crossover operator. For simplicity of illustration, we assume $p_c = 1$.

First we consider families 1 and 4 (fig. 2a, d). In family 1, we choose a founder randomly. Suppose ID 1 is chosen. This individual's mating partner, ID 2, is also a founder. We then swap these two individuals' haplotypes with those of the corresponding individuals of family 4. Retaining the haplotypes of all other founders in these two families, we determine new haplotypes of the nonfounders using the principles and procedures stated earlier. We now get the post-crossover families given in figures 3a, b.

Next, we consider families 3 and 4 (fig. 2c, d). Suppose the randomly chosen founder of family 3 is ID 4. The mating partner of ID 4 is ID 3, who is not a founder. We now consider two options. Option I: swap haplotypes of both ID 3 and ID 4 of family 3 with those of the corresponding individuals of family 4; and option II: swap haplotypes of ID 4 between families 3 and 4. Each option has a different implication. For option I, the s_i values of ID 3 will need to be adjusted, but swapping both ID 3 and ID 4 tantamounts, albeit implicitly, to swapping their offspring (say, N_0 in number) too (since the haplotypes of the offspring depend on the haplotypes of their parents only). Thus, adjustment of s_i values of ID 3 requires only the consideration of haplotypes of two individuals – parents of ID 3. For option II, on the other hand, when the haplotypes of only ID 4 are swapped, no adjustment of s_i values of ID 3 is necessary, but adjustment of s_i values of all N_0 offspring are necessary.

Since the idea underlying the crossover operator is to introduce 'genetic diversity' to some, but not too large, extent, we recommend that options I and II be chosen with probabilities $N_0/(N_0 + 2)$ and $2/(N_0 + 2)$, respectively. In the present illustration, options I and II are chosen with probabilities 2/3 and 1/3, respectively.

Suppose the randomly chosen founder of family 3 (fig. 2c) was indeed ID 4, and the randomized procedure stated above resulted in choice of option II. Then, after swapping with family 4 (fig. 2d), the resultant families will resemble figures 3c, d. Now s_i values of members need to be adjusted. For ID 3 of figure 3c, $s_1(1) = 1$, $s_1(2) = 2$, $s_1(3) = 1$, $s_2(1) = 2$, $s_2(2) = s_2(3) = 1$; for ID 5, $s_3(1) = s_3(2) = s_3(3) = 1$, $s_4(1) = s_4(2) = 2$, $s_4(3) = 1$. Hence, $X_{31} = X_{32} = X_{33} = 1$, $X_{34} = 0$, $X_{51} = X_{52} = X_{53} = 0$, $X_{54} = 1$. Therefore, $f(\mathbf{X}) = 4$. For ID 3 of figure 3d, $s_1(1) = s_1(2) = s_1(3) = 1$, $s_2(1) = s_2(2) = s_2(3) = 2$; for ID 5, $s_3(1) = s_3(2) = s_3(3) = 1$, $s_4(1) = s_4(2) = s_4(3) = 1$. Hence, $X_{31} = X_{32} = X_{33} = X_{34} = X_{51} = X_{52} = X_{53} = X_{54} = 0$. Therefore, $f(\mathbf{X}) = 0$.

Mutation Operator

In conventional GA, mutation is a simple random alteration of 0 to 1 and conversely. But in the present problem, we cannot perform the mutation operation randomly as it might lead to Mendelian incompatibility. So we need to modify the mutation operator appropriately. We propose two types of mutation. In type I mutation, we select a member randomly from the set of founders. We next choose a heterozygous locus randomly from that member and interchange the two alleles of that locus along with any necessary adjustment required to maintain Mendelian compatibility. The type II mutation is similar

to the conventional GA mutation operator. Here, we find out the locus at which a recombination has taken place in a particular candidate. We choose a locus randomly from the list of such loci, interchange the alleles at that locus and make the necessary adjustments required for this change. The type II mutation rate used is varied dynamically as:

$$p_m(f) = \frac{1}{2} \times \frac{f - f_{min}}{\bar{f} - f_{min}}, \text{ if } f \le \bar{f}$$

$$= \frac{1}{2}, \text{ otherwise;}$$

where f = number of recombinants for the particular candidate for which the mutation rate is to be calculated, $\bar{f}$ = average number of recombinants in the previous generation, and f_{min} = the minimum number of recombinants attained in the previous generation. The rate for type I mutation is taken to be 1/10 of this rate. This rate is consistent with the usual mutation rate adopted in conventional GA.

An illustration: To illustrate the mutation operator, we consider the post-crossover family depicted in figure 3c. To decide on the type(s) of the mutation operator to be applied, we choose two random numbers from Uniform[0,1] distribution. If the first random number is ≤ 0.5, then we apply the type I mutation; otherwise not. If the second random number is ≤ 0.5, then we apply the type II mutation; otherwise not. Suppose the random numbers were so chosen that only the type I mutation needs to be applied. To apply this mutation operator, we randomly choose a founder and a heterozygous locus of this founder. Suppose the random numbers are so chosen that locus 2 of ID 4 is selected. We then interchange alleles at this locus between the two chromosomes of ID 4 to get the new family of figure 4a. Then, for ID 3, $s_1(1) = s_1(2) = s_1(3) = 1, s_2(1) = s_2(2) = s_2(3) = 2$; for ID 5, $s_3(1) = 1, s_3(2) = s_3(3) = 2, s_4(1) = s_4(2) = s_4(3) = 2$. Hence, $X_{31} = X_{32} = X_{33} = X_{34} = 0, X_{51} = 1, X_{52} = X_{53} = X_{54} = 0$, yielding $f(\mathbf{X}) = 1$.

To illustrate the type II mutation operator, we consider the post-crossover family depicted in figure 3a. In this family, as we have noted earlier, there were 4 recombination events. These were: one between loci 1 and 2 and another between loci 2 and 3 in ID 1; one between loci 1 and 2 in ID 2; and, the fourth crossover was between loci 2 and 3 in ID 4. Therefore, type II mutation operator can be applied to any of the locus positions 1, 2 and 3 of ID 1; positions 1 and 2 of ID 2 and positions 2 and 3 of ID 4. We randomly choose any of these 7 possible positions. Suppose the position chosen is locus 3 of ID 4. Then, after application of the mutation operator, the family in figure 3c is transformed to the family depicted in figure 4b. In this family, for ID 3, $s_1(1) = 1, s_1(2) = 2, s_1(3) = 1, s_2(1) = 2, s_2(2) = s_2(3) = 1$; for ID 5, $s_3(1) = s_3(2) = s_3(3) = 1, s_4(1) = s_4(2) = s_4(3) = 1$. Hence, $X_{31} = X_{32} = X_{33} = 1, X_{34} = 0, X_{51} = X_{52} = X_{53} = X_{54} = 0$, yielding $f(\mathbf{X}) = 3$.

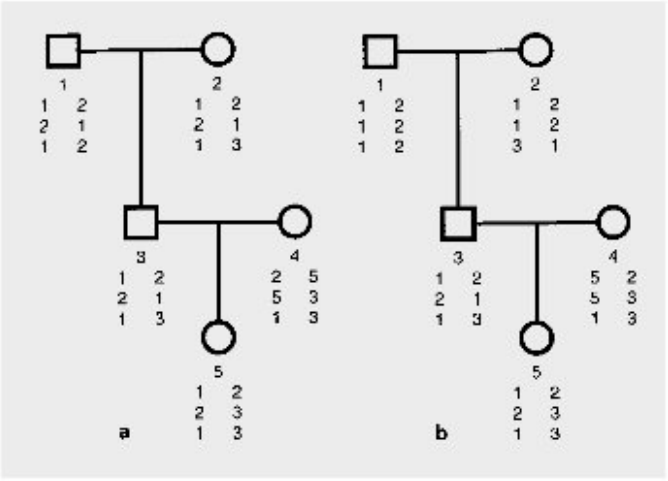


Fig. 4. Post-mutation example pedigrees. **a** Post type I mutation. **b** Post type II mutation.

An Overview of the Algorithm

The proposed genetic algorithm for haplotyping in pedigrees is an iterative scheme. Each iteration comprises several steps. At each step a specific operation is performed in the following sequence. In the first step, we randomly select many possible haplotypic configurations from the given genotypes of the pedigree. These form the first-generation candidates in our GA. In the second step, we apply the reproduction operator. In the third step, we apply the crossover operator. In the fourth step, we apply the mutation operator. In the fifth step, we examine whether all the candidates (families) have the same number of recombinants. If so, we have arrived at an optimal solution; else, we go back to the second step and repeat the steps till we obtain an optimal solution.

Results and Discussion

We have implemented the algorithm in a computer program, HAPLOPED, written in C, and have tested it successfully on numerous pedigrees of varying structures and with different numbers of loci. (HAPLOPED is available by writing to PPM and will be sent via electronic

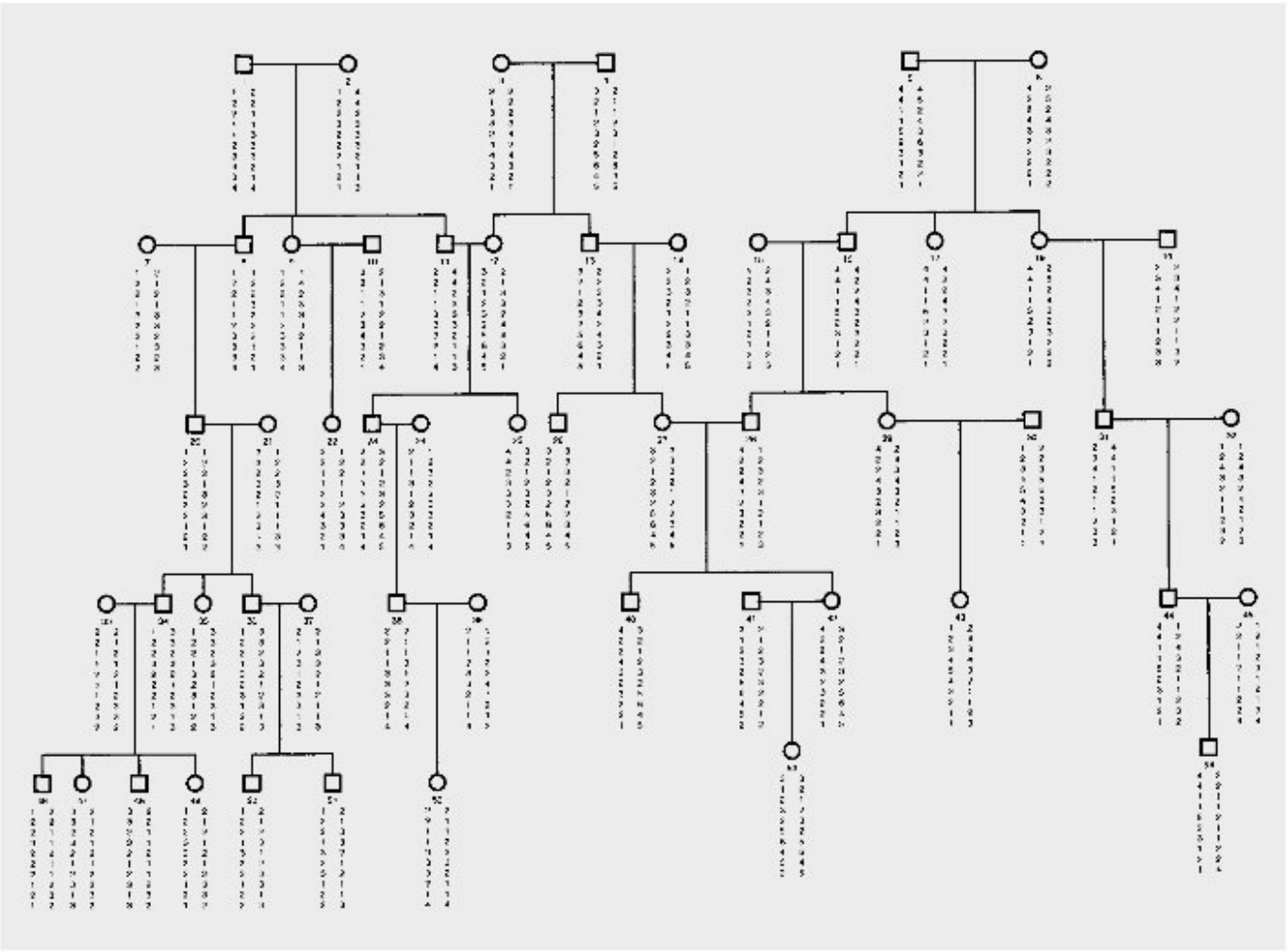


Fig. 5. Example pedigree used for illustrating the proposed genetic algorithm for haplotyping. Reconstructed haplotypes are depicted under each individual.

mail.) For brevity, we shall discuss the results based on one particular pedigree (fig. 5) comprising 54 members. Reconstructed haplotypes are presented in figure 5, although the input to HAPLOPED comprised only the genotypic information at each of the ten loci for every pedigree member. We have also artificially and randomly introduced recombinants in the pedigree and have analyzed the effects of having 1, 2 and 3 recombinants. In each case (0, 1, 2 or 3 recombinants), we have run HAPLOPED 100 times to estimate some essential benchmark parameters. In each run, we have used 160 ($\approx 3 \times 54$, per our recommendation, see below). The results are presented in table 3. The following salient features are evident from this table: (1) the estimated number of recombinants upon convergence of the algorithm generally equals the actual number of recombinants, provided that the number of replicate families is reasonably large, (2) the number of iterations to convergence generally increases with increase in the number of loci and actual number of recombinants, (3) the average CPU time (which is directly proportional to the average number of iterations to convergence) taken over 100 runs on a VAX-8750 machine running VMS Ver. 5.5-2 is reasonably short, and (4) the variance of the CPU time is also small.

In addition to the pedigree structure and genotypes of individuals, the only other input to HAPLOPED is N = the number of replicate families (candidate strings) to be considered in each generation. If N is small, the algorithm

Table 3. Results of 100 runs for obtaining haplotypes of members of the pedigree presented in figure 5 using HAPLOPED

True number of recombinants	Estimated number of recombinants	Avg. number of iterations to convergence	CPU time to convergence	
			mean	variance
0	0 (100)	67	243.31	21.64
1	1 (99) 2 (1)	113	387.49	36.08
2	2 (99) 3 (1)	145	453.62	49.57
3	3 (99) 4 (1)	205	668.04	60.65

Figures in parentheses are frequency of runs.

may converge to a local minimum, while if N is large, convergence to the global minimum is almost guaranteed but the computing time required will be considerably higher. Based on the results of haplotyping using HAPLOPED in a large number of pedigrees of different structures and with varying numbers of loci (results are not present for brevity), we suggest that N should be at least twice, preferably three times, the number of members in the given pedigree in order to ascertain a high degree of precision of the final haplotyping results.

We note that occasionally the algorithm may converge to an incorrect optimum value even when the number of replicate families used is large. In our experiments (fig. 5), in each of the rare cases when an extra recombinant was inferred by the algorithm, we have discovered that a founder was haplotyped incorrectly which, in turn, resulted in the incorrect inference of an extra recombinant. For example, among the 100 runs with true number of recombinants equalling 2, in the only run when an extra recombinant (at locus 6 in individual 13) was inferred, we found that haplotypes assigned to the founder individual 1 (a parent of 13) by the process of random assignment of alleles to chromosomes was different from the other runs, which incorrectly forced a recombination event in individual 13. Normally, such errors are expected to get corrected during the iterative process, but in rare instances (<1%) these do not get corrected.

It is, however, theoretically possible, albeit quite improbable for large pedigrees and/or several loci, that there are multiple haplotypic configurations with the same minimum number of recombinants. If there are multiple optima, then the proposed GA will find only one of these in a single run. However, multiple runs of the algorithm are expected to detect the multiple optima.

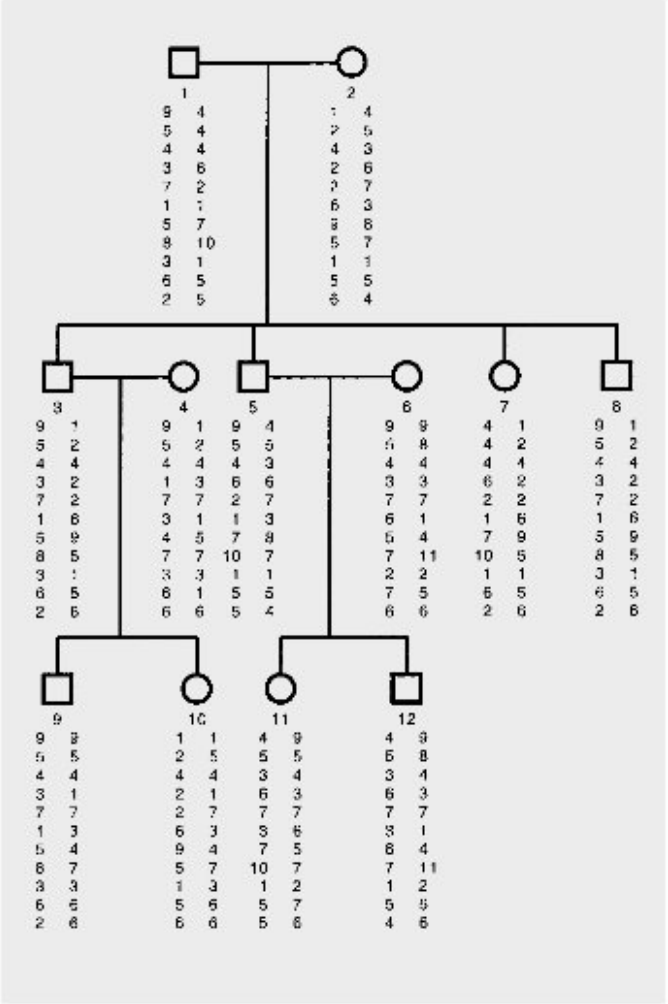
We have used HAPLOPED to determine haplotype configurations of the family that was used to exemplify haplotype reconstruction by GENE-HUNTER [7, fig. 9]. Figures 6a, b provide the haplotype reconstructions of this pedigree obtained in two separate runs of HAPLOPED. As is seen, figure 6a is exactly the same as figure 9 of Kruglyak et al. [7]. There are 4 recombinants in figure 6a – individuals 5 (paternal recombination between loci 2 and 3), 7 (paternal recombination between loci 9 and 10), 10 (maternal recombination between loci 1 and 2) and 11 (paternal recombination between loci 6 and 7). However, figure 6b presents an interesting deviation. In this set of reconstructed haplotypes there are 4 recombinants as well. However, in this case the recombinant individuals are: 5 (paternal recombination between loci 2 and 3), 7 (paternal recombination between loci 9 and 10), 9 (maternal recombination between loci 1 and 2) and 11 (paternal recombination between loci 6 and 7). The reason for the deviation in the set of reconstructed haplotypes in figure 6b is that in this particular run of the algorithm, the reconstructed haplotypes of individual 4, who is a founder, is different from the run based on which figure 6a was drawn. Thus, HAPLOPED provides multiple optimal solutions which is extremely desirable for purposes of genetic analyses especially when estimates of allele frequencies are not very reliable and/or assumptions of Hardy-Weinberg equilibrium and linkage equilibrium may not hold.

Comparison of the present algorithm and HAPLOPED with existing algorithms and programs (GENHUNTER, SIMWALK) for haplotyping is not possible because of the differences in assumptions and input data required by these different procedures. Existing computer programs for haplotyping use likelihood-based methods

that require as inputs locus-specific allele frequencies and pairwise recombination fraction estimates, in addition to those required by the present algorithm, which are pedigree structure, locus order and locus-specific genotypes of pedigree members. Further, unlike the present algorithm, the existing methods assume that each locus is in Hardy-Weinberg equilibrium. Thus, in experiments to compare HAPLOPED with other existing programs, differences in inferred haplotypes of members may not reflect relative efficiencies of the procedures, but may simply be due to differences in assumptions and input data. Comparisons of CPU time may also not provide a good measure of relative efficiency. The present algorithm (and HAPLOPED) belongs to a class of algorithms that is not comparable to the class of likelihood-based algorithms on which the existing haplotyping programs are based. We emphasize that the present algorithm is much less demanding in terms of data and assumptions, works fast and efficiently, and provides multiple optimal haplotype configurations of a pedigree, if multiple optima exist. We also recognize that in the class of GAs, the strategy suggested in the present study (e.g., defining GA individuals as pedigrees, etc.) may not be unique. That is, there may be other ways to implement a genetic algorithm for pedigree haplotyping; the present algorithm is only one in the GA class. Finally, we would like to point out that as in likelihood-based methods, missing data result in increase of computational time and complexity. We have not yet been able to solve the problem of missing data to our complete satisfaction and have, therefore, not yet incorporated any modifications in HAPLOPED for handling missing data.

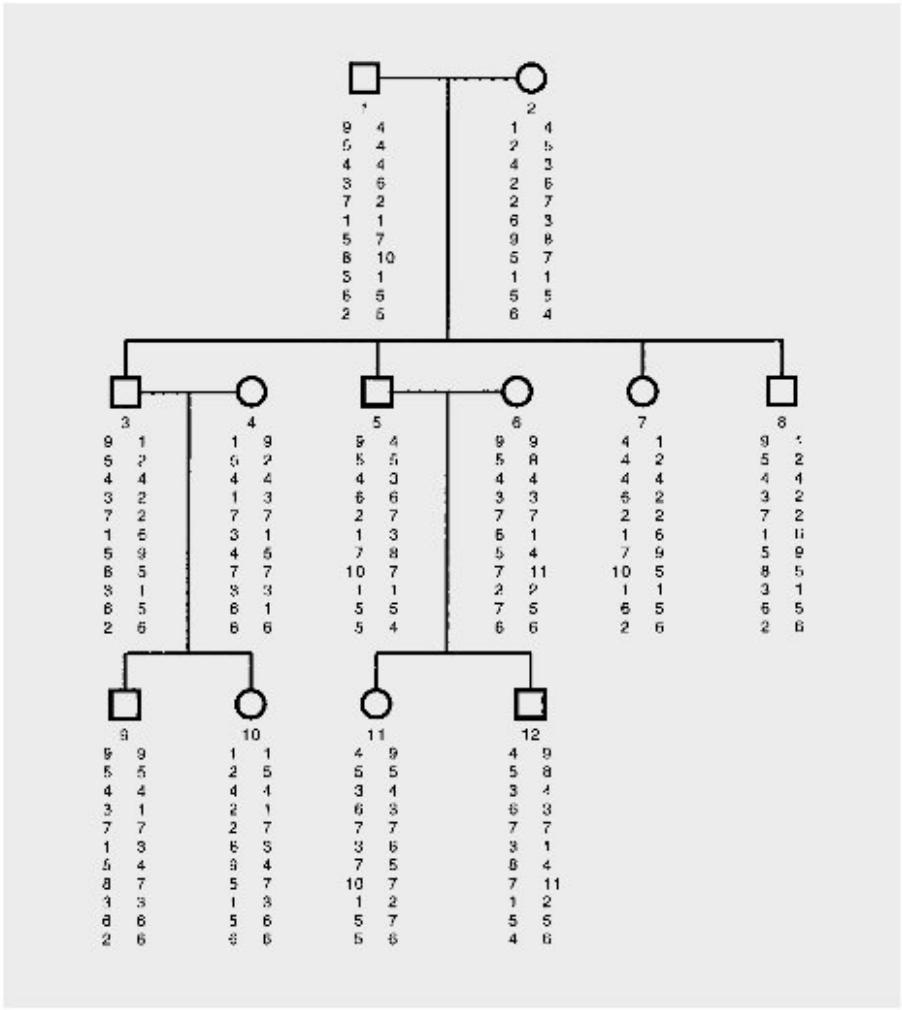
We are currently working on protocols for handling of missing data within the proposed framework and algorithm. We outline below the possible strategies we are considering for solving this problem and also provide some relevant comments. When genotype data at specific loci are missing for some individuals in the pedigree, our overall strategy is to logically impute the missing genotypes. Imputation of missing genotypes for individuals with relatives (parents/offspring/siblings) available in the pedigree is done by considering their (relatives') genotypes. Sometimes this leads to inference of missing genotypes with certainty. But more often, it only reduces the set of possible genotypes at those loci for which data are missing in such individuals. No reduction in the set of possible genotypes can be made if data are missing in founding individuals who have no relatives in the pedigree.

If missing genotypes are uniquely imputed, we need to deal with only fully genotyped pedigrees, which is



a

straightforward using our algorithm. No changes in prescribed values of any of the parameters (e.g., M) are necessary. However, as already mentioned, imputed missing genotypes are almost always nonunique; therefore, a pedigree with missing data almost invariably results in multiple 'imputed, fully-genotyped' pedigrees. If only one pedigree can be chosen from among these 'imputed, fully-genotyped' pedigrees which is in some sense optimal or most probable, then the complications arising out of missing data are completely solved. We are currently examining two strategies for making such an optimal choice. These are: whenever multiple imputed genotypes are possible at a locus for an individual (even after taking into consideration the genotypes of relatives), (1) assign only that genotype which is the most frequent at that locus in the population (if population data are available) or among



b

Fig. 6. a Example pedigree used in Kruglyak et al. [7]. Haplotypes depicted under each individual were reconstructed using the proposed algorithm which completely agrees with the haplotypes reconstructed using the algorithm given in Kruglyak et al. [7]. There are four recombinants in this pedigree. **b** Alternative haplotype reconstruction of the same family with the same genotypes of individuals using the proposed algorithm. There are four recombinants in this pedigree.

the set of founders in the pedigree; or (2) compute the posterior probabilities of the various possible genotypes using the genotype information of first-degree relatives and assign that genotype as the imputed one for which the posterior probability is the highest. Strategy (1) is computationally simpler than strategy (2); however, strategy (2) is statistically sounder (at least in the sense of local optimality). Either strategy yields a single, fully genotyped pedigree which can be haplotyped using the proposed algorithm. We are currently carrying out extensive simulations to examine the performance of these two strategies; results will be reported in a forthcoming publication. We emphasize that, whatever the results of the simulations, the proposed protocol for handling missing data will not require any change in the genetic algorithm for haplotyping.

Acknowledgments

We are grateful to Dr. N.R. Pal and Prof. Marc Feldman for many useful suggestions. This work was partially supported by a grant from the Department of Biotechnology, Government of India, to P.P.M.

References

1 Wijsman EM: A deductive method of haplotype analysis in pedigrees. Am J Hum Genet 1987;41:356–373.

2 Haines JL: Chromlook: An interactive program for error detection and mapping in reference linkage data. Genomics 1992;14:517–519.

3 Lange K, Matthysse M: Simulation of pedigree genotypes by random walks. Am J Hum Genet 1989;45:959–970.

4 Lange K, Sobel E: A random walk method for computing genetic location scores. Am J Hum Genet 1991;49:1320–1334.

5 Weeks DE, Sobel E, O'Connell JR, Lange K: Computer programs for multilocus haplotyping of general pedigrees. Am J Hum Genet 1995;56:1506–1507.

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

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

8 Sobel E, Lange K, O'Connell JR, Weeks DE: Haplotyping algorithms; in Speed TP, Waterman MS (eds): Genetic Mapping and DNA Sequencing. IMA Volumes in Mathematics and Its Applications (edited by Friedman A, Gulliver R). New York, Springer, 1995.

9 Davis L (ed): Genetic Algorithms and Simulated Annealing. London, Pitman, 1987.

10 Goldberg DE: Genetic Algorithms in Search, Optimization and Machine Learning. Reading, Addison Wesley, 1989.

11 DeJong KA: Adaptive system design: A genetic approach. IEEE Trans Syst Man Cyber 1980; 10:566–574.

12 Srinivas M, Patnaik LM: Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cyber 1994;24: 656–667.