Fast multipoint linkage analysis and the program Allegro

Daniel F. Gudbjartsson[1,2,4]
Kristjan Jonasson[1,4]
Michael L. Frigge[1]
Augustine Kong[1,3]
[1] Decode Genetics, Reykjavík, Iceland
[2] Institute of Statistics and Decision Sciences, Duke University, Durham, NC, USA
[3] Department of Human Genetics, University of Chicacgo, Chicago, IL, USA
[4] The first two authors have contributed equally to this work

Summary

Several improvements to computational algorithms for multipoint linkage analysis are developed. The new algorithms achieve considerable speed-up over previous ones, and allow larger families to be analyzed. The algorithms have been implemented in a computer program, Allegro. Allegro has the same basic functionality as the well known Genehunter program, includes the features of Genehunter-Plus, and contains some added functionality. Among the supported features are parametric LOD scores, allele sharing LOD scores, NPL scores and haplotyping. The program is simple to use and accepts the same data file format as Genehunter. It has been used extensively and typical speed-up compared to Genehunter is 30-fold.

The linkage analysis of Allegro and Genehunter involves three steps. Firstly, determination of single point probabilities of individual inheritance vectors, secondly multipoint calculation, where genotype information on neighboring markers is taken into account, and thirdly score calculation. The bulk of the time used by Genehunter involves steps 1 and 3 and it is here that the improvements are greatest; the key idea is to make use of tree traversal to avoid repeated calculation for similar inheritance vectors. In addition the fast Fourier transforms in step 2 have been made faster in the new algorithms. Finally, a new technique, founder couple reduction, halves the total execution time and memory requirement for many large pedigrees.

1 Introduction

The first generally available computer program for linkage analysis was Liped (Ott 1974; Ott 1976), which can calculate two-point *parametric LOD scores* for general pedigrees using the Elston-Stewart algorithm (Elston and Stewart 1971). Later programs, in particular Linkage (Lathorp et al. 1984; Terwilliger and Ott 1994) and Fastlink (Cottingham et al. 1993), provide multipoint parametric LOD-scores for a few markers, but the run time of these programs grows extremely rapidly with the number of markers. If the markers are polymorphic with several alleles each, the maximum number of markers these programs can practically handle is 3 or 4. Vitesse (O'Connell and Weeks 1995) adds 2 or 3 to the possible number of markers, but there are some restrictions on the pedigree structure. Other programs can handle special pedigrees, for instance Mapmaker/Sibs (Kruglyak and Lander 1995).

Multipoint linkage analysis for general pedigrees of moderate size, and involving many markers, was first made practical in 1996 with the introduction of the program Genehunter (Kruglyak et al. 1996). The run time of Genehunter grows exponentially with pedigree size (the pedigree in Figure 4 is close to the largest tractable), but only linearly with the number of markers. However, if there are no loops in the pedigrees, the run times of the linkage programs discussed in the previous paragraph grow linearly with pedigree size, so for them there is no practical size limit. With Genehunter, multipoint parametric LOD scores may be calculated, and in addition the sharing scoring functions S_{pairs} and S_{all} (Weeks and Lange 1988; Whittemore and Halpern 1994) may be used to calculate *nonparametric linkage* (NPL) scores. The algorithms of Genehunter use inheritance vectors, introduced by Lander and Green (1987) who also showed how to use *hidden Markov models* (HMM) to calculate probability distributions of inheritance vectors. The original version of Genehunter used a more or less direct approach to do the HMM step, but in version 1.2 this was replaced by a fast Fourier transform calculation (Kruglyak and Lander 1998).

Kong and Cox (1997) showed that NPL scores can sometimes be unacceptably conservative in the common case that the genotype data is incomplete. They suggested an alternative, *allele sharing* LOD scores, and created a modified version of Genehunter, Genehunter-Plus, to calculate them.

In this paper a number of improvements to the algorithms used in Genehunter are presented. A new computer program, *Allegro*, based on the new algorithms, has been written and is available from the authors. In addition to the calculation of parametric LOD scores, NPL scores and allele sharing LOD scores, Allegro reconstructs haplotypes in the same way as Genehunter and estimates the total number of recombinations in a similar way.

Due to the algorithmic improvements, Allegro runs much faster than Genehunter. Typically the speed-up is 20–40 fold, but may in some cases be as high as 100-fold. In addition, at a cost of 10–30% in run time, Allegro will, if necessary, use automatic recalculation and/or disk-swapping to cut down the memory requirements by a factor of 20–60 compared with Genehunter. However, the run time and memory requirements of Allegro are still exponential in the pedigree size. The pedigree in Figure 3 is close to the largest tractable with Allegro, but a little too large for Genehunter.

Allegro uses the fast Fourier transform (FFT) methodology of Kruglyak and Lander (1998) to calculate multipoint probabilities of individual inheritance vectors. It employs the founder reduction described by Kruglyak et al. (1996), reducing the size of inheritance vectors by one bit for each founder in the pedigree using a symmetry between the two alleles of the founder. The computational improvements fall in three categories. Firstly fast tree traversal is used in the calculation of single locus probabilities, and in the calculation of both parametric and non-parametric scoring functions, often speeding computation by two orders of magnitude. Secondly, some classical computational techniques have been used to speed the FFT by a factor of three or four. Thirdly, symmetry between founder couples is used to further reduce the size of inheritance vectors and gain additional speed.

1.1 Notation

The *full inheritance vector*, v, for a pedigree with m non-founders is an element of Z_2^{2m}. In other words, v is a bit-vector with $2m$ binary bits, two bits for each non-founder in the pedigree, a paternal bit and a maternal bit. For a given locus, the paternal bit expresses whether the paternally transmitted allele at the locus stems from the grandfather (if the bit is 0) or the grandmother (if the bit is 1), and similarly for the maternal bit. In the main paper on the Genehunter program (Kruglyak et al. 1996) it was shown how one may instead work with *reduced* inheritance vectors, having $n = 2m - f$ bits where f is the number of founders. For each founder, the corresponding bit of his or her first child is set to 0 and not expressed in the reduced vector. The number of possible (reduced) inheritance vectors is $N = 2^n$. Genotype information is assumed to be available at M marker locations.

The probability of a random variable taking the value x is denoted with $p(x)$ and $p(x|\cdot)$ denotes conditional probability, so that if g is the genotype data at all loci and v is an inheritance vector, then $p(v|g)$ is the probability of v given g. Similarly, if g_k is the available genotype data at locus k, then $p(v|g_k)$ denotes the probability of the vector v given the data at the locus. This probability, the *single locus probability*, is also denoted with $q_k(v)$. If the locus is understood, the subscript k is sometimes replaced with a family index, i, or dropped completely.

2 Fast tree traversal

The basic structure of the algorithms presented in Kruglyak et al. (1996), and implemented in the Genehunter program, is to loop over inheritance vectors in the outermost loop and over people in the pedigree in an inner loop. This applies to the calculation of single locus probabilities, parametric likelihood and non-parametric scoring functions (S_{pairs}, S_{all}). The drawback of this approach is that for each pair of inheritance vectors that only differ for branches of the pedigree, part of the calculation for the two vectors will be duplicated.

The gist of the method presented here involves changing the order of looping, and thereby avoiding these repeated calculations. The innermost sections of the algorithms consist of looping over inheritance vectors and people in the pedigree hand in hand. This idea is elaborated in section 2.1 where the calculation of the S_{pairs} scoring function is taken as an example. Thereafter the fast calculation of single locus probabilities is considered. The calculation of S_{all} and parametric likelihood is then discussed in section 2.3.

2.1 Example: Sharing in pairs

Assume that for each inheritance vector v a quantity $s(v)$ is wanted. The basic idea is to traverse the pedigree from the top down, in fact almost in the same order as the pedigree came into existence in reality. When a child is born (i.e., a person is added to the pedigree) there are generally four possibilities of assigning inheritance vector bits, depending on the results of the underlying meioses. If (some of) the bits are set to 0 and hidden, there are fewer possibilities (one or two). For each possibility, the inheritance vector is appropriately updated and the branch of the pedigree starting with the child is descended. Thus in general four new iteration branches are created, but fewer if some of the bits are hidden. A template algorithm may be written up for this idea. Let v be the inheritance vector corresponding to the part of the pedigree that has already been traversed, b be the bit to be added to the vector, and v^+ the updated vector. Define a collection of data, $\mathcal{D}(v)$, with the properties that it is cheap to calculate $\mathcal{D}(v^+)$ from $\mathcal{D}(v)$ and that $s(v)$ is readily calculable from $\mathcal{D}(v)$. The core of the method is a recursive algorithm:

addbit(v, $\mathcal{D}$, b):
  for b = 0, 1 do
    set $v^+ = (v, b)$ and calculate $\mathcal{D}^+ = \mathcal{D}(v^+)$
    if there are more bits, *addbit*(v^+, $\mathcal{D}^+$, next bit), else $\mathcal{D}^+$ contains data for $s(v^+)$.

The calculation is started off by executing *addbit* with an empty inheritance vector and $\mathcal{D}$ appropriately initialized. Note that if the calculation of $\mathcal{D}^+$ and s are both $O(1)$ then the total time complexity of the calculation is $O(N)$.

As an example, consider the calculation of the S_{pairs} scoring function (Weeks and Lange 1988; Whittemore and Halpern 1994). Let $S_{pq}(v)$ be the IBD sharing of two people p and q given v (i.e., $S_{pq}(v) = \sum_{i=0}^{1} \sum_{j=0}^{1} \phi_{ij}(p,q)$ where $\phi_{ij}(p,q)$ is 1 if allele i of p and allele j of q are IBD and 0 otherwise). Then

$$S_{\text{pairs}}(v) = \sum_{\substack{(p,q) \text{ is a pair} \\ \text{of affecteds}}} S_{pq}(v). \tag{1}$$

Let the founder alleles be numbered in an analogous way to the founder nodes shown in Figure 1. Let k_i be the number of times founder allele i turns up among the affected (given v), let s be the value of S_{pairs} for the portion of the pedigree already traversed and define $\mathcal{D} = (s, k_1, k_2, \ldots, k_{2f})$. When a bit of an unaffected child or the first bit of a child with two unhidden bits is added, $\mathcal{D}^+ = \mathcal{D}$. When the second or only bit of an affected child is added, let i and j be the sources of the alleles among the founders and obtain $\mathcal{D}^+$ with

$$\begin{aligned} s^+ &\leftarrow s + k_i + k_j \\ k_i^+ &\leftarrow k_i + 1 \\ k_j^+ &\leftarrow k_j + 1. \end{aligned}$$

Initially $\mathcal{D}$ must be set to reflect the IBD sharing among the founders and the children that have both bits hidden. The allele sources of a child are easily obtained from the allele sources of the parents, using the inheritance vector bits of the child, an $O(1)$ operation. The calculation of $\mathcal{D}^+$ is also $O(1)$ so the overall calculation will be $O(N)$.

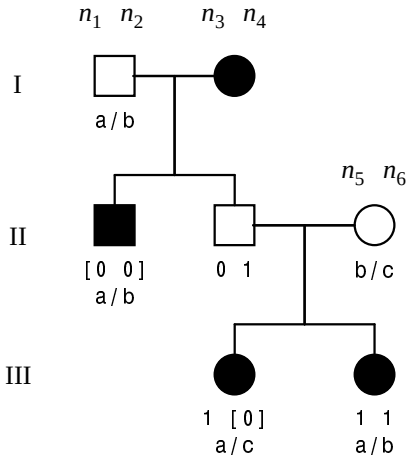

Figure 1: Example pedigree. The bits of the inheritance vector and available genotypes are below each individual and $n_1, ..., n_6$ are *founder nodes* (see text). Hidden inheritance vector bits are in square brackets.

As an example, consider the pedigree in Figure 1 with the inheritance vector shown. The pedigree members are identified by generation and position within the generation; for example II_3 is the third person in the second generation, counting from left. The reduced vector is $v = (0, 1, 1, 1, 1)$, and initially $s = 1$, $k_1 = 1, k_3 = 2, k_4 = 1$ and the rest of the k_i's are 0. When the bit of III_1 is added (with value 1), $i = 4$, $j = 5$, s is given the value $1 + 1 + 0 = 2$ and k_4 and k_5 are incremented to 2 and 1

respectively. When the second bit of III_2 is added (with values of both bits 1, as shown in the figure), $i = 4$, $j = 6$, and s becomes 2 + 1 + 0 = 3, the value of $S_{\text{pairs}}(v)$.

The procedure given above may be compared with the Genehunter calculation of S_{pairs}, which is: For each inheritance vector and each pair of affected people, calculate S_{pq} and add to S_{pairs}. The time required for this is $O(Na^2)$ where a is the number of affected. In addition Genehunter determines for each inheritance vector the source among the founders of each non-founder allele, an $O(Nm)$ calculation.

2.2 Single locus probability calculation

This section treats the calculation of the probability of each inheritance vector at a locus, given genotype data g_k at the locus (i.e., the calculation of $q_k(v)$ for all v). The $2f$ possible allele sources among the founders are referred to as *founder nodes* and the population allele frequency of an allele a is denoted with π_a. By Bayes theorem and because the inheritance vectors are all equally likely unconditionally,

$$p(v|g_k) = \frac{p(g_k|v)p(v)}{p(g_k)} \propto p(g_k|v), \tag{2}$$

where $\propto$ means "is proportional to". Consider the example in Figure 1, where the founder nodes are shown as $n_1, n_2, ..., n_6$. The reduced inheritance vector shown is (0, 1, 1, 1, 1), and with this vector, the sources of the alleles of II_1 are n_1 and n_3, the alleles of II_2 come from n_1 and n_4, those of III_1 come from n_4 and n_5 and those of III_2 from n_4 and n_6. With the genotype data given in the figure, there are two possible allele assignments to the founder nodes, (a, b, b, a, c, b) and (b, a, a, a, c, b) (in the first case, for example, the notation means that a is assigned to n_1, b to n_2 etc.). Thus the probability of observing this data is

$$p = \pi_a^2\pi_b^3\pi_c + \pi_a^3\pi_b^2\pi_c. \tag{3}$$

In general the probability is given by

$$p(g_k|v) = \sum_{a\in\mathcal{P}} \prod_{i=1}^{2f} \pi_{a_i} \tag{4}$$

where $\mathcal{P}$ is the set of possible allele assignments $a = (a_1, ..., a_{2f})$ to $(n_1, ..., n_{2f})$. By (2) the probability distribution of the inheritance vectors is found by normalizing the numbers given by (4). Because genotyped founders will for every inheritance vector contribute the same factor to each term, the product in (4) may in fact be taken over just the nodes of ungenotyped founders. The factor will be cancelled out in the normalization. In the example above, I_1 and II_3 always contribute $\pi_a\pi_b\pi_b\pi_c$ to each term.

The probabilities given by (4) may be calculated for each v by traversing the pedigree as in the template algorithm of section 2.1. At each stage in the algorithm the founder nodes are classified into three disjoint sets, *assigned nodes*, *unassigned nodes* and *edge nodes* ($\mathcal{A}$, $\mathcal{U}$ and $\mathcal{E}$). Each node in $\mathcal{A}$ has been assigned an allele but the nodes in $\mathcal{U}$ have not. $\mathcal{E}$ consists of pairs of nodes, each pair is joined with an edge which is labelled with two distinct alleles (e.g. a/b), implying two possible allele assignments to the node pair (a, b and b, a). If $i \in \mathcal{A}$, denote the allele assigned to i with a_i.

Initially, $\mathcal{E}$ consists of the nodes of the genotyped founders, with an edge joining the two nodes of each, labelled with his or her genotype, $\mathcal{U}$ consists of the rest of the founder nodes and $\mathcal{A}$ is empty. The set $\mathcal{D}$ consists of this node classification and associated allele assignment. When bits of an ungenotyped person are added to v, $\mathcal{D}$ is unchanged. Also, set $q(v)$ to 0 to start with.

Now assume that a person genotyped a/b is added to the pedigree and the associated unhidden bits are added to v. Even if one or both bits are hidden, it is still necessary to go through the following calculation, but looping over possible values of hidden bits is unnecessary so the total time complexity of the total calculation does not increase appreciably. Consider first the homozygous case, $a \neq b$. The values of the inheritance vector bits determine the sources of the alleles of the person among the founder nodes. Let e be an edge joining these nodes. There are 6 possible cases for the placement of the edge e

with respect to the sets $\mathcal{A}$, $\mathcal{U}$ and $\mathcal{E}$, indicated by the 6 edges in Figure 2, where these cases are labelled with circled numbers.

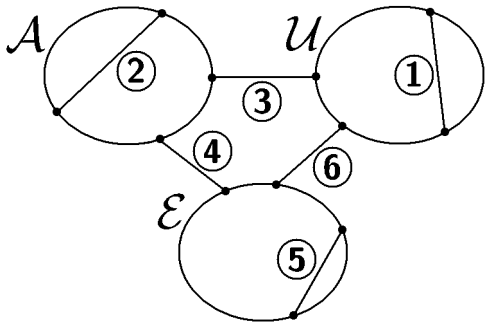

Figure 2: Classification of founder nodes and possibilities for adding genotyped persons.

In case 1 both ends of e are in $\mathcal{U}$ and they are moved to $\mathcal{E}$ along with the edge itself labelled with a/b. In cases 2, 3 and 4, let $e = (i, j)$ with $i \in \mathcal{A}$. Now the labelling of e is either consistent with the current allele assignment or not. If it is inconsistent the pedigree need not be descended any further with the current values of the bits, as the probability of all inheritance vectors with the current bit values is 0. In case 2 it is only necessary to check whether the alleles assigned to the ends of e are a and b (i.e., whether $\{a, b\} = \{a_i, a_j\}$). If they are, the traversing may continue with $\mathcal{D}$ unchanged, otherwise the assignment is inconsistent. In case 3, simply check whether a_i is one of a and b. If it is, assign the other one to j and move j from $\mathcal{U}$ to $\mathcal{A}$, otherwise the labelling is inconsistent. In case 4, check first if a_i is one of a and b and if this holds, check whether the other one is consistent with the labelling of the edge f in $\mathcal{E}$ adjacent to j. If this also holds, the allele assignment to both end nodes of f is forced and they are moved from $\mathcal{E}$ to $\mathcal{A}$. Otherwise the labelling is inconsistent. In cases 5 and 6 it may be necessary to introduce an extra loop in the algorithm. Let $e = (i, j)$ and first set $a_i = a$ and $a_j = b$. If this assignment is consistent, move i, j, and end nodes of adjacent edges in $\mathcal{E}$ to $\mathcal{A}$ and continue traversing. Then try $a_i = b$ and $a_j = a$, and if consistent traverse with the resulting assignment.

The homozygous case ($a = b$) is very similar to the heterozygous one, and the same 6 cases need to be considered. Some of the differences are that in case 1 the ends of e are moved to $\mathcal{A}$ instead of to $\mathcal{E}$ and in cases 5 and 6 it is never necessary to introduce an extra loop. Note that if there is inbreeding in the pedigree, it is possible that e joins a node with itself, when both alleles come from the same node.

When the traversing reaches a leaf of the pedigree, v is a particular inheritance vector and the set of founder nodes together with the set $\mathcal{E}$ and the allele assignments will be called the *founder graph* of v.

For the probability calculation a product over the edges associated with $\mathcal{E}$ is needed. Let $\mathcal{E}$ be the set of these and let the alleles labelling $e \in \mathcal{E}$ be a_e and b_e. Each time the last bit of the inheritance vector has been added to v, $q(v)$ is updated by adding to it:

$$\prod_{i \in \mathcal{A}} \pi_{a_i} \prod_{e \in E} 2\pi_{a_e} \pi_{b_e}. \tag{5}$$

At the end of the traversing q is normalized. As in (4) it is all right to take the products in (5) over just the nodes of ungenotyped founders.

As an example, consider the steps of the algorithm leading to the inheritance vector shown in Figure 1. Let a_i denote the allele assigned to n_i. Initially $\mathcal{A} = \emptyset$, $\mathcal{U} = \{n_3, n_4\}$ and $\mathcal{E} = \{n_1, n_2, n_5, n_6\}$ with edges labelled a/b and b/c joining n_1 and n_2 on one hand and n_5 and n_6 on the other. Adding the bits of II_1 gives rise to case 6, and a loop over the two possible assignments $(a_1, a_2, a_3) = (a, b, a)$ and (b, a, b) is generated. For both assignments $\mathcal{A} = \{n_1, n_2, n_3\}$, $\mathcal{U} = \{n_4\}$, and $\mathcal{E} = \{n_5, n_6\}$ with (n_5, n_6) labelled b/c. Adding the bits of III_1 again gives rise to case 6, with $e = (n_4, n_5)$ labelled a/c.

This time the assignment $(a_4, a_5) = (c, a)$ is inconsistent and it is only necessary to consider $a_4 = a$, $a_5 = c$ which is consistent and enforces $a_6 = b$. Thus when the bits of the last person, III_2, are added $\mathcal{A}$ consists of all the nodes with $\mathcal{U}$ and $\mathcal{E}$ empty. Adding these bits therefore corresponds to case 2 with $e = (a_4, a_6)$ labelled a/b and this is consistent with both the generated loops. In the first loop $\pi_a\pi_b\pi_a\pi_a\pi_c\pi_b$ is added to $q(v)$ and in the second loop $\pi_a\pi_b\pi_a\pi_a\pi_c\pi_b$ is added, in accordance with (3) (when looping only over ungenotyped founders, $\pi_a\pi_b$ and π_a^2 are added).

2.3 Other applications: Peeling and set sharing

Given phenotype and genotype data for a pedigree, the parametric LOD score at a given locus is defined as $\log_{10}\Lambda$ where Λ is the likelihood ratio,

$$\Lambda = \frac{p(d|g)}{p(d)} = \frac{\sum_v p(v|g)p(d|v)}{\frac{1}{N}\sum_v p(d|v)},$$

v is an inheritance vector, d is the available phenotype (disease status) data and g is the available genotype data (at all loci). The model is based on a single disease locus with two possible alleles, C and c. The penetrances and population allele frequencies are assumed known. The most efficient way to calculate $p(d|v)$ is to use a peeling algorithm as explained in the Genehunter paper (Kruglyak et al. 1996), in effect a version of the Elston-Stewart algorithm (Elston and Stewart, 1971). Here it is again possible to make use of recursive traversal to speed up the calculation, and this has been done in the implementation discussed in section 5 below. The implementation involves two types of peeling operations, (a) peeling all the children and one parent on to the other parent, and (b) peeling both parents and all children but one onto the remaining child. Loops are dealt with by breaking them as in Lange and Elston (1975). The traversal order is not the same as in the previous section. Instead a suitable peeling order is decided on, and this then determines the traversal order. The initial loop is over inheritance vector bits of children in the first peeling operation, the next loop is over bits of children in the second operation and so on. Thus the first operation is only performed once for all inheritance vectors that have the bits of the children in the first family set in the same way, the second operation is performed once for the vectors that have the bits of the first two families set in the same way, and so on.

Consider the example in Figure 1. Suppose that II_3, III_1 and III_2 are peeled onto II_2 first. This will, for each possible setting of the last 3 bits in the reduced inheritance vector, involve the calculation of four numbers associated with II_2, namely the probabilities of his genotype being CC, Cc, cC and cc given the inheritance vector and the fact that his children are affected. His wife and children will not enter the rest of the calculation.

The set sharing scoring function introduced by Whittemore and Halpern (1994) is defined as

$$S_{\text{all}}(v) = 2^{-a}\sum_h \prod_{i=1}^{2f} b_i(h)! \tag{6}$$

where a is the number of affected individuals, h is a selection that picks one of the two alleles of each affected, $b_i(h)$ is the number of times founder allele i appears in h (given v), and the sum is taken over the 2^a possible ways of choosing h. The computational complexity of (6) is clearly large. Taken at face value it requires approximately $2f2^a$ multiplications for each inheritance vector, or about $f2^{a+2m-f+1}$ multiplications for them all. Using a slightly modified version of the algorithm in section 2.1 the total complexity can be brought down to around 2^{a+2m-f} multiplications.

3 Fourier transform and transition

Multipoint calculations are performed using a hidden Markov model (HMM, a good reference is Rabiner 1989) as proposed by Lander and Green (1987). At a marker locus k, let $l_k(v)$ denote the probability of

an inheritance vector v given the genotype data at all loci to the left of k (i.e. loci i with $i < k$) and let $r_k(v)$ denote the probability of v given the genotype data to the right of k. In the following the product xy of two vectors denotes the vector with i-th component $x_i y_i$, and the difference between bit-vectors is taken modulo 2, so that $x - y$ is 1 in positions where x and y differ and 0 where they are the same. Let the recombination fraction between two adjacent loci, γ and γ', be θ, and let $T_\theta \in R^{2^n}$ denote the transition probabilities between them, so that $p(\text{vector is } v \text{ at } \gamma \mid \text{vector is } w \text{ at } \gamma') = T_\theta(v - w)$. Then $T_\theta(u) = \theta^{|u|}(1-\theta)^{n-|u|}$, where $|u|$ denotes the weight of u (the number of 1 bits in u). Bayes theorem implies that the probability distribution of the vectors at locus k, given the data to the left of k and at k, is proportional to $l_k q_k$ and therefore $l_{k+1} \propto (l_k q_k) * T_{\theta_k}$, where θ_k is the distance between markers k and $k+1$ and $*$ is convolution,

$$(f * g)(x) = \sum_v f(x - v) g(v). \tag{7}$$

In the same way, $r_k \propto (r_{k+1} q_{k+1}) * T_{\theta_k}$. The probability distribution of the inheritance vectors at k given genotype data g at all loci may then be obtained through $p(\cdot|g) \propto l_k q_k r_k(v)$.

A way of calculating the convolution (7) using a fast Fourier transform (FFT) is given by Kruglyak and Lander (1998). The Fourier transform $\hat{f}$ of f is defined through

$$\hat{f}(w) = \sum_v (-1)^{|vw|} f(v),$$

and the inverse transform is obtained through $\hat{f} = 2^n \check{f}$. From this it follows that $f * g = (\hat{f}\hat{g})^{\vee}$. At face value the calculation of $f * g$ using (7) requires 2^{2n} multiplications and the same number of additions. However with the FFT $\hat{f}$ may be calculated using $n2^n$ additions and no multiplications (see section 3.1), and furthermore Kruglyak and Lander give a closed form for $\hat{T}_\theta$. Thus the total calculation time for (7) is brought down to about 2^n multiplications and $2n2^n$ additions.

Note the important property, that the transition probability from v to w only depends on the difference $w - v$. That the Fourier transform is applicaple in this calculation, depends on this fact.

3.1 Faster FFT

The FFT algorithm for $\hat{f}$ described by Kruglyak and Lander (1998) and used in the Genehunter program is to set $f_0 = f$ and then calculating for i = 1, 2, ..., n:

$$f_i(v) = (-1)^{v_i} f_{i-1}(v) + f_{i-1}(v - e_i) \qquad (\text{for all } v \text{ in } Z_2^n) \tag{8}$$

where e_i is the i-th unit vector and f_i overwrites f_{i-1} in memory. After this calculation f_n contains $\hat{f}$. The number of additions for the calculation is $n2^n$. However, on modern computers with several CPU registers and cache memory, the time required for the calculation may not simply be governed by $n2^n$. Accessing the registers is fastest, accessing the cache is slower and the main memory is slowest. If the program calculates (8) by looping over all possible values of v, and n is so large that f_i does not fit into the cache all at once (typically for n larger than about 13) frequent swapping of data in and out of the cache will result. To reduce the necessary swapping, one may recursively make use of the property of the Fourier transform that if f is partitioned into two equally sized subvectors, $f = (g, h)$, then $\hat{f} = (\hat{g} + \hat{h}, \hat{g} - \hat{h})$. Now partition g and h similarly and so on, until each subvector fits into the cache. Then use (8) to find the transform of the subvectors. Note that because the size of f is a power of 2, the subvectors in each partitioning can be made equal in size.

The direct implementation of (8) involves nested loops, i is looped over in the outer loop and K in the inner one. To take advantage of the fast CPU registers, (8) may be implemented by "unrolling" these loops. An example of unrolling is to rewrite the program fragment "*for* $i = 1, ..., 2$, *for* $j = 1, ..., i$, *let* $a_{ij} \leftarrow 2a_{i-1,j}$" as "$a_{11} \leftarrow a_{01}$; $a_{21} \leftarrow a_{11}$; $a_{22} \leftarrow a_{12}$". An extra bonus may be that unrolling eliminates the bookkeeping cost of loops. The number of floating point registers in modern computers

is typically 16 or 32, and it turns out that for larger n the original not unrolled version of (8) is faster than the unrolled version, possibly because then the f_i do not all fit into the registers. The optimal value of the largest n for which unrolling will pay off is somewhat different between computers.

A final rather trivial improvement is to replace multiplication by 2 that enters into the implementation of (8) by the left shift operator <<.

To assess the speed-up attainable by these modifications of the FFT algorithm, timing tests were carried out for several different vector sizes on four different platforms. Loops were unrolled for all n less than a given parameter, MAXUNLOOP and the tests included trying out different values of this parameter. The platforms tried were: (1) Sun Ultra Enterprise 450 running Solaris using the Gnu C++ compiler, (2) the same computer using the native CC compiler, (3) 200 MHz Dec Alpha with Gnu C++, and (4) 500 MHz Pentium III running Linux with Gnu C++. For 12 bit vectors the average speed-up was by a factor in the range 2.4–3.3 (2.8 on average), for 15, 18 and 21 bits by a factor in the range 2.8–5.3 (average 3.5), and for 24 bits the speed-up was somewhat lower, in the range 1.9–2.7 (average 2.4). These values include the saving obtained through using recursion (25–75%), loop unrolling (25–75%), and the left shift operator (5–10%) and are based on choosing MAXUNLOOP as the value that was most often optimal on each platform (3 for both compilers on the Sun, 2 for the Pentium and 4 for the Dec Alpha).

4 Founder couple reduction

The founder reduction discussed above and employed in Genehunter results in considerable savings in computer time, often by several orders of magnitude. For each founder in the pedigree the time required is halved. A brief overview of the mathematics behind the founder reduction is given here, followed by a description of the new founder couple reduction and the mathematical complications it introduces.

The founder reduction is based on the symmetry that exists between the two alleles/haplotypes of a founder. More specifically, let v be a configuration of the inheritance vector without founder reduction. Suppose, for a male (female) founder, that v' is obtained by adding 1 (mod 2) to all the paternal (maternal) bits of his (her) offspring. The two configurations v and v' are indistinguishable based on the observed genotype data, i.e., $p(v'|\text{genotype data}) = p(v|\text{genotype data})$, and can be grouped into an equivalence class. One relevant feature here is that the equivalence classes again form a group under addition (mod 2). Notice that for any two configurations v and w, the four cases, $(v + w)$, $(v + w')$, $(v' + w)$ and $(v' + w')$, all belong to the same equivalence class. Indeed, $(v + w) = (v' + w')$, and $(v + w') = (v' + w) = (v + w)'$. As a result, a step in the HMM calculation (e.g., the calculation of l_{k+1} from $l_k q_k$) corresponds to a convolution, and the Fourier transform approach will continue to work. Kruglyak and Lander (1998) show how to calculate the Fourier transform of the transition probabilities for the equivalence classes.

Consider a pedigree which has a couple who are both founders (founder couple), have at least one grandchild, are both not genotyped and do not have children with other mates. The symmetry between the members of this couple can be used to further reduce the amount of computation and memory requirements by approximately a factor of two. Specifically, for each grandchild of the founder couple, define the 'corresponding bit' as the paternal bit if the father of the grandchild is the offspring of the founder couple, and define the 'corresponding bit' as the maternal bit if the mother of the grandchild is the offspring of the founder couple. Let v be some configuration of the inheritance vector, and let v^* be like v with two modifications: (1) 1 is added (mod 2) to the corresponding bit of each of the grandchildren of the founder couple, and (2) the paternal bit and maternal bit of each child of the founder couple are switched. If the founder couple are not genotyped, then the data cannot distinguish between v and v^*. Hence v and v^* can again be grouped into an equivalence class as in the case of the founder reduction.

There is, however, one complication. The equivalence classes so created do not form a group under addition (mod 2) as illustrated by the following example. Focus on 3 bits where the first bit is the

corresponding bit of a grandchild of the founder couple, and the second and third bits are the paternal and maternal bits of an offspring of the founder couple. Let $v = (1, 0, 0)$ and $w = (1, 0, 1)$. Then $(v + w) = (0, 0, 1)$ and $(v + w)^* = (1, 1, 0)$, but $(v^* + w^*) = (0, 0, 0) + (0, 1, 0) = (0, 1, 0)$. Since $(v^* + w^*)$ is neither equal to $(v + w)$ nor $(v + w)^*$, $(v + w)$ and $(v^* + w^*)$ do not belong to the same equivalence class. Because of this complication, the HMM calculation of the multipoint probabilities of the equivalence classes no longer correspond to a convolution, and hence one cannot apply the theory in Kruglyak and Lander (1998) directly to handle the Fourier transforms. Our solution to this problem is to perform two separate convolutions, instead of one, and then taking the sum. The computational time can still be halved, because the ingredients that have to be prepared for performing the two convolutions are essentially the same. The details are left to the interested reader. The technicalities may be deduced from the source code of the program.

With the founder reduction and the founder couple reduction, the effective number of bits for both computational time and memory requirements is $2m - f - c$ where c is the number of founder couples satisfying the stated conditions.

5 Implementation: Allegro

The methods described in this paper have been incorporated in a computer program for multipoint linkage analysis, Allegro, which is available through the internet site http://www.decode.is/allegro and the email adress allegro@decode.is. As stated in the introduction, Allegro has the same basic functionality as Genehunter, but also includes the functionality of Genehunter-Plus. Thus Allegro calculates multipoint parametric LOD scores, NPL scores, allele sharing LOD scores, reconstruction of haplotypes, estimated recombination count between markers (observed map), p-values for NPL scores (based on perfect data approximation), and entropy information. The X chromosome is supported in all calculations.

The major advantage of Allegro over Genehunter/Genehunter-Plus is its speed and reduced memory requirement. In addition Allegro offers some functional improvements, as detailed in the following subsections. The add-ons include additional scoring functions, alternative information measure, simulation of the crossover process under the null (i.e. no linkage) given all the genotype data, and simulation of multipoint genotype data under the alternative of a single gene parametric disease model given the phenotypes.

The main specific limit in Allegro is that the size of inheritance vectors after bit reduction must not exceed 31 bits. There are no specific limits on the number of markers or founders whereas Genehunter has a limit of 50 markers and 16 non-founders.

One further advantage of Allegro is that multiple analyses may be performed together at little extra cost. For example it can in one run fit several different parametric models, use several different scoring functions and/or family weighting schemes, or compute both single point and multipoint LOD. Genehunter is limited to one parametric and one non-parametric model per run.

Allegro has been tested extensively at Decode Genetics. It has been used to analyze several thousand pedigrees, the output has been compared with that from other linkage analysis programs (Genehunter and Linkage) and a fair amount of effort has been put into eliminating programming errors.

5.1 Statistical inference

For multipoint linkage analyses, Allegro supports both classical parametric models, and the allele sharing models described in Kong and Cox (1997). For compatibility with Genehunter it also calculates NPL scores (as defined by Kruglyak et al., 1996). Compared to the parametric LOD scores, the definition and interpretation of the allele sharing LOD scores are less well known, and so a brief review is provided here. The scoring functions and weighting schemes that Allegro provides for allele-sharing LOD scores will also be described.

In the following, subscript i denotes dependence on the i-th family, and the notation '$\sum_{v_i} ...$' means summing over all configurations of the inheritance vector of family i. Given a non-parametric scoring function such as S_{pairs} or S_{all}, defined in (1) and (6), let S_i be the score for family i and Z_i be the standardized form of S_i, i.e., $Z_i = (S_i - \mu_i)/\sigma_i$ where μ_i and σ_i are the mean and standard deviation of S_i under the null hypothesis of no linkage. For any locus, the NPL score is defined as

$$\text{NPL} = \frac{\sum_i \gamma_i \bar{Z}_i}{\sqrt{\sum_i \gamma_i^2}}$$

where $\bar{Z}_i$ is the expectation of Z_i conditional on the genotype data, i.e., $\bar{Z}_i = \sum_{v_i} p(v_i|g_i) Z_i(v_i)$ (as before $p(\cdot|g_i)$ denotes the distribution of inheritance vectors in the i-th family and at the locus of interest, conditional on all the genotype data on the family, and under the null hypothesis of no linkage). The γ_i's are family specific weights which Genehunter sets to be equal for all families. When the genotype data provide complete IBD information so that $\bar{Z}_i = Z_i$, the NPL score has mean 0 and variance 1 under the null hypothesis. To obtain p-values, one can either use a stardard normal approximation or evaluate the exact distribution of the NPL score assuming complete IBD information. But, when the IBD information is incomplete, these ways of calculating p-values tend to be conservative, because the variance of the NPL score is now smaller than 1 under the null hypothesis.

To resolve the problem of the conservativeness, Kong and Cox (1997) proposed two models, a *linear model* and an *exponential model*, which are both closely related to the NPL score. In the linear model the probability of an inheritance vector v_i for the i-th family, at the locus of interest, is modelled as

$$p(v_i|\delta) \quad = \quad c_i \left\{ 1 + \frac{\delta \gamma_i [S_i(v_i) - \mu_i]}{\sigma_i} \right\}.$$

Here c_i is $1/N_i$ (the probability of v_i under the null hypothesis of no linkage). The linear parameter δ can be thought of as measuring the size of the genetic effect and is 0 under the null. The exponential model is formulated as

$$p(v_i|\delta) \quad = \quad c_i r_i(\delta) \exp \left\{ \frac{\delta \gamma_i [S_i(v_i) - \mu_i]}{\sigma_i} \right\}$$

where $r_i(\delta)$ is an appropriate renormalization constant that ensures $\sum_{v_i} p(v_i|\delta) = 1$ (e.g., $r_i(0) = 1$). Again δ is 0 under the null and large for a large genetic effect. For both the linear model and the exponential model, the LOD score is defined as

$$\text{LOD} = \log_{10} \frac{p(g|\hat{\delta})}{p(g|\delta = 0)} = \sum_i \log_{10} \frac{p(g_i|\hat{\delta})}{p(g_i|\delta = 0)}$$

where $\hat{\delta}$ is the maximizer of $p(g|\hat{\delta})$. Computationally, one takes advantage of the property that the distribution of g_i depends on δ only through v_i, which means that $p(g_i|v_i, \delta) = p(g_i|v_i, \delta = 0)$. Also recall that $p(g_i|v_i)$ denotes the conditional probability assuming no linkage, so that $p(g_i|v_i, \delta = 0) = p(g_i|v_i)$. It follows that

$$\begin{aligned} p(g_i|\delta) \quad &= \quad \sum_{v_i} p(g_i, v_i|\delta) = \sum_{v_i} p(g_i|v_i, \delta) p(v_i|\delta) = \sum_{v_i} p(g_i|v_i, \delta = 0) p(v_i|\delta) \\ &= \quad \sum_{v_i} \frac{p(g_i|\delta = 0)}{p(v_i|\delta = 0)} p(v_i|g_i, \delta = 0) p(v_i|\delta) \\ &= \quad N_i p(g_i|\delta = 0) \sum_{v_i} p(v_i|g_i) p(v_i|\delta) \end{aligned}$$

and

$$\begin{aligned}
\text{LOD} &= \sum_i \log_{10} \frac{\sum_{v_i} p(v_i|g_i)p(v_i|\hat{\delta})}{\sum_{v_i} p(v_i|g_i)p(v_i|\delta=0)} \\
&= \sum_i \log_{10} \frac{\sum_{v_i} p(v_i|g_i)p(v_i|\hat{\delta})}{\sum_{v_i} p(v_i|g_i)(1/N_i)} \\
&= \sum_i \log_{10} \sum_{v_i} p(v_i|g_i)p(v_i|\hat{\delta}) + \sum_i \log_{10} N_i.
\end{aligned}$$

For the linear model, it is shown by Kong and Cox that the LOD score can be further simplified to

$$\text{LOD} = \sum_i \log_{10}(1 + \hat{\delta}\gamma_i \bar{Z}_i).$$

For both the linear and exponential models, the LOD score corresponds to a standard likelihood ratio statistic, i.e., $2\ln 10 \cdot \text{LOD}$ has asymptotically a chi-square distribution with one degree of freedom under the null. Indeed, since the sign of $\hat{\delta}$ is relevant, define $Z_{\text{lr}} = \text{sign}(\hat{\delta})\sqrt{(2\ln 10 \cdot \text{LOD})}$. Asymptotically, under the null, Z_{lr} has a standard normal distribution, and approximate p-values can be computed based on that. For both models, $\delta > 0$ corresponds to excess sharing ($E(Z_i) > 0$) while negative δ corresponds to deficit sharing ($E(Z_i) < 0$). Hence, when testing for linkage, it is natural to focus on positive values of δ. For example, Kong and Cox define the LOD score by maximizing only over positive values of δ. In Allegro δ is maximized over both positive and negative values for the following reasons: (1) while a substantial LOD score associated with a negative $\hat{\delta}$ is not evidence for linkage, it may suggest errors in the data, (2) a locus with a negative $\hat{\delta}$ value is even more unlikely to be a susceptibility locus than a locus with $\hat{\delta}$ around zero, (3) by allowing $\hat{\delta}$ to be negative, there is a direct correspondence between Z_{lr} and the NPL score, i.e., they should be close to each other for large samples if the IBD information is complete, and (4) some measures of information (see below) require that $\hat{\delta}$ be allowed to be negative. Observe that it is very important to notice the sign of δ when interpreting a LOD score. Indeed, when the results are displayed, plotting either $\text{sign}(\hat{\delta}) \cdot \text{LOD}$ or Z_{lr} is recommended.

When δ is small, the linear and exponential models are very similar. With large samples and complex diseases this is likely to be the case, and the LOD scores of the two types of model are likely to be very similar. Also, if the IBD information is complete, then the NPL score is the classical efficient score statistic for both models, which means that the test based on the NPL score will be very similar to the tests based on the LOD scores of the two models. Indeed, when the IBD information is complete, there is a one to one correspondence between the NPL score and the Z_{lr} score for the exponential model, which means that the tests based on the two statistics are formally equivalent. However, normal approximation tends to work better with Z_{lr} than the NPL score (see Nicolae et al. 1998). Badner et al. (1998) compared NPL scores with the LOD scores computed with the linear model and found the latter to have more power and to be better calibrated in all cases tested. In their study, the NPL score is so conservative that the power is sometimes even below the nominal Type I error. This is because they perform two-point analyses and the information is very far from complete.

As a default, we recommend the exponential model over the linear model, although the former is not inherently superior. As noted, the exponential model is more closely related to the NPL score. Also, normal approximation can work better for the Z_{lr} of the exponential model than the Z_{lr} of the linear model when there are a small number of families, or the families are very different in size. In this sense, the exponential model is more robust.

The main output files of Allegro for allele sharing contain the values of $\hat{\delta}$, the LOD score, Z_{lr}, and the NPL score. For equal weighting of the families ($\gamma_i = 1$ for all i), the NPL score computed by Allegro is exactly the same as the NPL score which Genehunter calculates. The files also contain a measure of the information contained in the data at each locus, both for individual families, and for all the families combined. The combined information measure is given by equation (2.2) in Nicolae (1999);

it is always in the interval $[0, 1]$ and is 1 if the information on the S_i's is complete (i.e., if the genotype data uniquely determines the sharing between affecteds). It is more difficult to define a good per-family information measure; a reasonable choice is the one given by equation (2.4) in Nicolae (1999), and this is what Allegro uses. One of the drawbacks of this measure is that it is not guaranteed to be positive. In addition to these measures, Allegro can calculate the entropy information I_E of Kruglyak et al. (1996), as Genehunter does. Note that because of the founder couple reduction the values of I_E calculated by Allegro can be different from those calculated by Genehunter. For example, if there are data on a cousin-pair, but not on their parents, and a highly informative marker implies that the cousin-pair shares one allele with probability close to one, then $I_E \approx 1/2$ for Genehunter, but $I_E \approx 2/3$ for Allegro. If perfectly informative genotypes are also available for all four parents of the cousins, then $I_E = 3/4$ for Genehunter, but for Allegro $I_E = 1$. This example highlights the fact that the information measure produced by Genehunter can be substantially below 1 even when there is no more information to be gained regarding allele sharing of the affecteds, and further genotyping can be a waste of resources.

Allegro supports five scoring functions, S_{pairs}, S_{all}, S_{robdom}, S_{mnallele} and S_{homoz}. S_{mnallele} is minus the statistic (A) of Sobel and Lange (1996) and is called $S_{-\#\text{alleles}}$ in McPeek (1999). It is designed to be appropriate for a recessive disease. S_{robdom} was introduced by McPeek (1999) and is designed to be appropriate for a rare dominant disease with some phenocopies. S_{homoz} is like S_{pairs}, except that when there is inbreeding and an individual is homozygous IBD he may contribute to the scoring. McPeek (1999) discusses the different scoring functions, and in summary she recommends S_{pairs} as a compromise choice for a scoring function that performs well over all disease models. She finds S_{robdom} to be slightly more powerful than S_{all} (and easier to calculate) in the dominant cases, and recommends S_{mnallele} for recessive diseases.

Allegro can accept the family weights γ_i in three different ways: (1) all weights are set to 1 (equal weights, corresponding to the weighting scheme of Genehunter), (2) for a given s, the weights are set to σ_i^s for all i, and (3) the weights are specified for each family individually. Option (2) is most meaningful when used with the scoring function S_{pairs}. Here, choosing $s > 0$ corresponds to putting more weight on larger families (with more affecteds). Choosing $s = 1$ means the S_i, instead of the Z_i, are weighted equally and corresponds to weighting all affected pairs equally. Lange (1997, pp. 96–97) discusses different weighting schemes and suggests using a generalization of the scheme proposed by Hodge (1984), as a compromise between weighting families equally (option (1) or $s = 0$) and weighting with $s = 1$. One way of obtaining such a compromise is to set $s = 1/2$, which currently is our preferred default, and should be quite close to the weighting scheme suggested by Lange. Option (3) provides the user with complete flexibility, and can be used to perform the type of conditional analysis described in Cox et al. (1999).

5.2 Haplotyping and Simulation

Haplotyping is an important ingredient in locating disease genes, and as stated above, Allegro reconstructs haplotypes approximately. Define an *inheritance vector path* to be a set of inheritance vectors, one at each locus. The haplotype reconstruction is obtained by first calculating the most likely inheritance vector path, using the Viterbi algorithm (Viterbi 1967), which is an application of dynamic programming to hidden Markov models. Having obtained this, the most likely allele assignment for each vector in the path is found using the corresponding founder graphs (see section 2.2), thus producing the desired haplotype. The haplotype produced is the same as that given by Genehunter, but the algorithm in Allegro takes advantage of both the founder reduction and the founder couple reduction, whereas Genehunter works with full inheritance vectors. This is the primary reason for the large speed-up of the Allegro haplotyping compared with that of Genehunter, exhibited in the next subsection. The program Simwalk2 (Sobel et al. 1996) uses the same modelling for haplotyping but the first step (i.e., finding the most likely inheritance vector path) is solved approximately. Note that this type of modelling is not guaranteed to find the most likely haplotype, but the result should be reasonably close in most cases.

Allegro offers two types of simulation, the crossover process may be reconstructed by simulation given the genotypes (assuming no linkage with a disease), and genotypes may be simulated given phenotypes and a single gene parametric disease model. The first type of simulation essentially involves drawing a random inheritance vector path from the conditional distribution of such paths (conditional on the genotype data). This random draw uses the probabilities q_k and l_k (see section 3). The second type of simulation is carried out in three stages. In the first stage peeling is used to obtain the distribution of inheritance vectors at the disease locus given the disease model and the phenotypes. The second stage draws an inheritance vector from this distribution and the third stage involves generating all the genotype data by simulating the crossover process in both directions from the locus.

5.3 Observed Map

As Genehunter, Allegro computes an 'observed' map, but this map is defined slightly differently from that of Genehunter. For two adjacent markers, Genehunter first calculates the 'observed' recombination fraction

$$\hat{\theta} = \frac{E(\text{number of recombinations}|g, \text{input map})}{\text{number of meioses}}$$

where g denotes all the genotype data. Then the Haldane map function is used to convert $\hat{\theta}$ to the 'observed' genetic distance, i.e.,

$$\hat{x} = -\frac{1}{2}\ln(1 - 2\hat{\theta}).$$

By contrast, our 'observed' genetic distance is defined directly as

$$\tilde{x} = \frac{E(\text{number of crossovers}|g, \text{input map})}{\text{number of meioses}}.$$

The units of both $\hat{x}$ and $\tilde{x}$ are in Morgans; multiplying by 100 converts to centiMorgans (cM). Also, for both Genehunter and Allegro, $\hat{x}$ and $\tilde{x}$ for the whole chromosome are calculated by summing over the $\hat{x}$'s and $\tilde{x}$'s of adjacent marker pairs.

The difference between $\hat{x}$ and $\tilde{x}$ is subtle and tends to be small when they are calculated for the combined data of many families, but the difference can be substantial for an individual family. Consider a family consisting of a sib pair and the two parents. When the markers are perfectly informative, it is possible to deduce whether the total number of recombinations of the two paternal meioses combined is odd (one) or even (zero or two), and similarly for the maternal meioses. But there is no direct evidence on whether a particular meiosis has a recombination. For two adjacent markers, suppose two recombinations are observed, one in the two paternal meioses and one in the two maternal meioses, then $\hat{\theta}$ is 2/4 and $\hat{x}$ is formally infinity. In this situation, Genehunter sets $\hat{x}$, somewhat arbitrarily, to be 999.9 cM. If the two adjacent markers are specified as 10 cM apart by the input map, then $\tilde{x}$ = 50.7 cM.

For two markers that are 10 cM apart, the chance of having two recombinations, one for the paternal meioses, and one for the maternal meioses, is 0.027. However, given 31 markers on a chromosome, 10 cM apart, the chance of this happening for at least one pair of adjacent markers is over 50%. It means that, if the markers are all completely informative, then $\hat{x}$ for the whole chromosome will be infinity (or 999.9 cM with the Genehunter truncation) for more than one half of the sib pairs. Also, two different sib pairs which have the same number of recombinations over the same chromosome can have very different values of $\hat{x}$ if one pair happens to have recombinations for both the maternal and paternal meioses in the same marker interval, and one pair does not. By contrast, $\tilde{x}$ for the chromosome will be the same for these two sib pairs. In general, we feel that $\tilde{x}$ is more appropriate when the 'observed' map is used to assess whether an individual family has an unusual number of recombinations/crossovers, which may result from genotyping errors or misspecified relationships. Also worth noting is that the expectation of $\tilde{x}$ is exactly the actual genetic distance if the input map is correct, and $\tilde{x}$ for the families combined is a weighted average (weighted by the number of meioses) of the $\tilde{x}$'s of the individual families, properties that are obviously not shared by $\hat{x}$.

Finally, note that it makes sense to focus only on meioses which one can potentially have information on recombinations/crossovers. Specifically, if a founder has a single child, then the data cannot provide any crossover information regarding that meiosis. With Allegro, these meioses are left out for the calculation of $\tilde{x}$ in both the numerator and denominator. Genehunter also attempts to do that. However, it is not consistent in how it treats the individual families and the families combined. The most curious thing is that in the calculation of $\hat{\theta}$ for the families combined, the numerator includes these noninformative meioses while the denominator does not.

5.4 Run time experimentation

Allegro has been timed against version 1.3 of Genehunter (timing trials indicate that the more recent version 2 of Genehunter is in fact slower than version 1.3), and the results for four different pedigrees are summarized in Table 1. The pedigree F24 is shown in Figure 3, F18 is obtained from this by removing individuals III_7, III_8 and III_{11} (i.e. individuals 7, 8 and 11 in the third generation, counting from the left), F12 is obtained by further removing III_3, IV_2 and IV_4, and FLoop is shown in Figure 4. Artificial sample data (based on real data) for 50 markers was analyzed and NPL scores as well as parametric LOD scores were calculated at markers and at one locus midway between each pair of markers. The run times for Allegro include the time needed for allele sharing LOD scores. To obtain these with Genehunter requires the modified version Genehunter-Plus, but the run time increase is negligible. In the test data there are 2.2% missing genotypes (for the genotyped individuals) in F24, 2.0% missing in F18, 1.4% in F12 and in FLoop 4.8% of the genotypes are missing.

Table 1 gives results of the run time experimentation on a Sun Ultra Enterprise 450 computer with 2 Gb of memory. Each run consists of one parametric and one nonparametric analysis for 50 markers. Notice the halving of execution time when the founder couple reduction is used for F12, F18 and F24, and that this saving is fourfold for FLoop which has two founder couples. Recall that to make this reduction possible, the members of the founder couples must not be genotyped. Theoretically the time complexity of the FFT, which is the dominating calculation, is of the order of $n2^n$ where n is the bit size. The run times shown in the table roughly confirm this. Note that disk swapping and recalculation slow the F24 calculation somewhat. Run times for 14, 16, 20 and 22 bit pedigrees, also obtained by deletion of ungenotyped leaves from F24, were observed to be in line with the numbers given in Table 1. Genehunter cannot handle a pedigree larger than F18 for 50 markers on this computer.

Timing experiments were also carried out on a 500 MHz Pentium III computer running Linux. Somewhat disappointingly the speed-up factors of Allegro compared with Genehunter turned out to be lower than on the Sun. The average of all the factors in Table 1 is 44.8, but the corresponding average for the Pentium is 33.8.

Table 2 shows the proportion of the total time spent on individual steps in the multipoint analysis of the pedigrees F18 and FLoop, both for Allegro and Genehunter. The computation is the same as for Table 1, without haplotyping and with founder couple reduction. The figures for Genehunter are for versions 1.2 or 1.3, which have identical computational engines. The numbers for both programs were obtained by running a profiling program, and then confirmed by multiple runs, choosing in the control files different analyses to be performed in each run. The breakdown is (obviously) more accurate for Allegro, but should be fairly accurate for Genehunter also. The large speed-up observed for the single point calculation and peeling (for parametric score) is effected by the fast tree traversal methodology and the founder couple reduction (which doubles the speed for F18 and quadruples it for FLoop). The speed-up of the HMM step is due to the faster FFT, the founder couple reduction, and some additional differences between the two programs.

Kruglyak and Lander (1998) state that the HMM step dominates the overall calculation of NPL scores (and LOD-scores for loop-less pedigrees) in all versions of Genehunter existing at the time (versions 1.0, 1.1 and 1.2). Table 2 provides evidence that this is incorrect, and that instead it is the single point calculation that dominates. For Allegro, on the other hand, the HMM calculation dominates, and

it is clear that speed improvements here would be most important.

6 Concluding remarks

Future research and/or improvements to the Allegro program could include the incorporation of quantitative trait analysis, a way of dealing with interference in the crossover process, handling of different recombination rates for the two sexes, and a robust way of dealing with errors that unavoidably exist in the data (including both errors in the genotype data and the marker map).

It would be important if robust approximation techniques, that would enable larger pedigrees to be analysed, could be developed. One would also like to be able to deal with the fact that founders may not be independent (i.e. unrelated) neither within nor between pedigrees. It may be possible to estimate how related a pair of founders is, and make use of this, even if it is either not possible or feasible to use the detailed pedigree of their relatedness.

Acknowledgements

We thank the statistics group at Decode Genetics for extensive testing of Allegro.

References

Badner JA, Gershon ES, Goldin LR (1998) Detection of common disease alleles. Am J Hum Genet 63:880-888

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

Cox N, Frigge M, Nicolae DL, Concannon P, Hanis CL, Bell GI, Kong A (1999) Loci on chromosomes 2 (NIDDM1) and 15 interact to increase susceptibility to type 2 diabetes in Mexican Americans. Nat Genet 21:213–215

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

Hodge SE (1984) The information contained in multiple sibling pairs. Genet Epidemiol 1:109-122

Kong A, Cox NJ (1997) Allele-sharing models: LOD scores and accurate linkage tests. Am J Hum Genet 61:1179–1188

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

Kruglyak L, Lander ES (1995) Complete multipoint sib-pair analysis of qualitative and quantitative traits. Am J Hum Genet 57:439–454

Kruglyak L, Lander ES (1998) Faster multipoint linkage analysis using Fourier transforms. J Comput Biol 5:1–7

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

Lange K (1997) Mathematical and statistcal methods for genetic analysis. Springer-Verlag, New York

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

Lathrop GM, Lalouel JM, Julier C, Ott J (1984) Strategies for multilocus linkage analysis in humans. Proc Natl Acad Sci USA 81:3443-3446

McPeek MS (1999) Optimal allele-sharing statistics for genetic mapping using affected relatives. Genet Epidemiol 16:225-249

Nicolae DL (1999) Allele sharing models in gene mapping: a likelihood approach. PhD thesis, Department of Statistics, University of Chicago

Nicolae DL, Frigge ML, Cox NJ and Kong A (1998) Discussion to "Multipoint linkage analysis using affected relative pairs and partially informative markers" by Siegmund D and Teng J. Biometrics 54:1271-1274

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

Ott J (1974) Estimation of the recombination fraction in human pedigrees: efficient computation of the likelihood for human linkage studies. Am J Hum Genet, 26:588–597

Ott J (1976) A computer program for linkage analysis of general human pedigrees. Am J Hum Genet 28:528–529

Rabiner LR (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77:257–286

Sobel E, Lange K (1996) Descent graphs in pedigree analysis: applications to haplotyping, location scores, and marker-sharing statistics. Am J Hum Genet 58:1323–1337

Sobel E, Lange K, O'Connel JR, Weeks DE (1996) Haplotyping algorithms. In: Speed T, Waterman MS (eds) Genetic mapping and DNA sequencing. The IMA volumes in mathemtics and its applications 81. Springer-Verlag, New York

Terwilliger JD, Ott J (1994) Analysis of human genetic linkage, Johns Hopkins University Press, Baltimore and London

Viterbi AJ (1967) Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. IEEE Trans Informat Theory IT-13:260–269

Whittemore AS, Halpern J (1994) A class of tests for linkage using affected pedigree members. Biometrics 50:118–127

Figure legends

Figure 3. Pedigree F24 used in run time experimentation. The bit size is 24 or 25 depending on whether founder couple reduction is used or not. Filled in individuals are affected, the others are treated as having unknown affection status, and genotypes are not available for individuals that are crossed over. The pedigrees F18 and F12 are obtained from F24 by deleting unaffected leaves. See main text for more details.

Figure 4. Pedigree FLoop, containing a simple marriage loop, used in run time experimentation. If the two founder couples are used to reduce the bit size this is a 17 bit pedigree; if not the bit count is 19. Filled in individuals are affected and genotypes are unavailable for crossed over individuals.