Efficient Haplotyping for Families

by

Amy Lynne Williams

B.S., Computer Science, University of Utah (2003)
B.S., Mathematics, University of Utah (2003)
S.M., Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2005)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2010

Author . . . .
Department of Electrical Engineering and Computer Science
October 16, 2009

Certified by . . . .
David K. Gifford
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Terry P. Orlando
Chairman, Department Committee on Graduate Theses
Electrical Engineering and Computer Science

Efficient Haplotyping for Families

by

Amy Lynne Williams

Submitted to the Department of Electrical Engineering and Computer Science
on October 16, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

Hapi is a novel dynamic programming algorithm for haplotyping nuclear families that outperforms contemporary family-based haplotyping algorithms. Haplotypes are useful for mapping and identifying genes which cause and contribute to the etiology of human disease, and for analyzing the products of meiosis to locate recombinations, enabling the identification of recombination hotspots and gene conversions. They can also be used to study population history, including expansion, contraction, and migration patterns in humans and other species. Hapi's efficiency is a result of eliminating or ignoring states and state transitions that are unnecessary for computing haplotypes. When applied to a dataset containing 103 families, Hapi performs over 3.8–320 times faster than state-of-the-art algorithms. These efficiency gains are practically important as they enable Hapi to haplotype family datasets which current algorithms are either unable to handle or are impractical for because of time constraints. Hapi infers both minimum-recombinant and maximum likelihood haplotypes, and because it applies to related individuals, the haplotypes it infers are highly accurate over large genomic distances.

Thesis Supervisor: David K. Gifford
Title: Professor of Electrical Engineering and Computer Science

Acknowledgements

Special thanks are due to Professors David Gifford, David Housman, and Martin Rinard, who served as advisors for this work. Although my interests were somewhat divergent from their areas of focus—and particularly from Martin Rinard's—they provided me freedom and an atmosphere that allowed me to pursue these ideas. Each contributed in significant ways to this process and helped shape the focus of my work.

I am grateful beyond words for the opportunity I have had of coming to graduate school. I have learned more than I imagined and in more contexts than I anticipated. I'm thankful to Wilson Hsieh for encouraging me to attend and also for his patient guidance in the writing of my bachelor's thesis. His tutelage of my writing helped me learn to enjoy something I had previously loathed, and helped foster a much-needed skill.

My dear sister Elizabeth—"Elizaboo"—has been an invaluable support through all of this. Many late nights spent coding or writing were warmer because of our playful discussions. She is the best sister I ever had, and I mean that!

Angelina Lee was a wonderful officemate and we had many great talks over the years—both in the office and outside. I am extremely grateful to her willingness to listen to sometimes very detailed explanations of research ideas that were far flung from her area of interest. She helped me think through things.

I am grateful for the music that has been the accompaniment to my work these several years. The song of life would be incomplete without music to give it voice. A dance, a vision—a dream of futurity—plays out in the minds and hearts of one moved by symphonic poetry. Such dreams kept me journeying on through the arduous and perplexing tasks that constitute the Ph.D.

Many friends from Church helped and I couldn't begin to list them all. I do wish to give special thanks to Dave Alba, Teppo Jouttenus, Sylvia Pearce, and Rebecca Sansom. Each provided me with support throughout this journey and I'm grateful to call them friends. Sylvia's feedback on a few bits of writing were

I cannot begin to state—nor could I ever overstate—God's power in leading me to this point in my life. His Almighty hand has most clearly brought me to where I am now. I am nothing without Him. I'm grateful for what lies ahead.

God's plan is not the plan of pleasure;
it is the "plan of happiness."
—Neal A. Maxwell

Contents

List of Figures

Chapter 1

Introduction

Recent advances in high throughput genotyping technologies are making possible previously unknown, large-scale studies in the areas of medical genetics, basic biology, and human history. Datasets with information about hundreds of thousands of genetic variants for thousands of human subjects are now common and continue to be gathered. Additionally, whole-genome sequencing methods are becoming less expensive and therefore more attractive for research study datasets. Whole-genome sequence data contain a wealth of information, including genotypes for all genetic variants in an individual.

Genotype data provide information about the genetic variants in an individual that are present at specific locations on both genome copies—one copy inherited from each parent. However, genotype data do not identify which of the variants at multiple genotyped locations reside on the same homologous chromosome copy. A genotype can therefore be thought of as an unordered pair of variants since either variant could reside on either chromosome copy. An *allele* is a designation, often a letter, for a particular variant form that resides at location, so a genotype is made up of two alleles. A *locus* (plural, *loci*) is a specific location where a particular genetic element, including a gene, variant, etc., occurs. Absent any outside information, any combination of alleles across a series of genotyped loci may appear on the same chromosome copy.

A *haplotype* is an assignment of each genotype allele to the homologous chromosome copy it resides on and is useful in many contexts. Haplotypes can greatly increase the statistical power of genome wide association studies and thereby aid in identifying genes involved in human diseases. This increase in statistical power is particularly relevant to the study of human diseases with complex genetic contributions to their etiology. As well, haplotypes are applicable to the study of the results of meiosis—within a single generation or averaged across many generations—providing the opportunity to detect patterns such as recombination hotspots or gene conversions.

Besides medical and biological applications, haplotypes are useful in estimating the age of a mutation event in a population. In individuals that carry a mutation, the length of shared DNA surrounding that mutation in those individuals is a reflection of the age of the mutation in the population. Longer lengths of DNA indicate that less

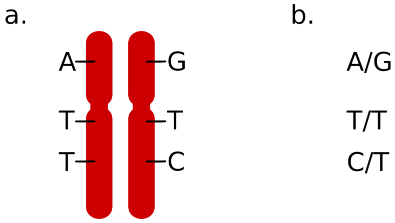


Figure 1-1: (a) Homologous chromosomes labeled with one of the two possible haplotypes for (b) three SNP genotypes on the same chromosome.

time has transpired since the mutation was first introduced into the population. Thus, by comparing lengths of identical segment across a set of individuals' haplotypes (i.e., *haplotype blocks*), one can study human history, including expansion, contraction, and migration patterns.

Figure 1-1 shows three genotypes that correspond to two possible haplotypes, one of which is shown. The second possible haplotype for the genotypes in the figure has the A allele of the first locus on the same chromosome as the C allele of the third locus. Genotypes are written with a slash between the two allele values. In this example two of the three genotypes are *heterozygous*, meaning their two alleles differ. The second genotype is *homozygous*, i.e., it contains two copies of the same allele.

In general, for n heterozygous genotypes on a chromosome, there are 2^{n-1} possible haplotypes. When the two chromosome copies are differentiated by delineating which copy came from each parent, the number of possible haplotypes is 2^n. In the Figure, we might differentiate the chromosomes by saying that the that the left homolog came from the father and the right from the mother.

The genotypes in Figure 1-1 are for single nucleotide polymorphisms, which are abbreviated SNPs and pronounced *snips*. At present, most genotyping assays target this form of genetic variants, but there are several other common forms of genetic variants that occur and that can be assayed. A SNP occurs at a known genomic position at which the resident nucleotide varies, i.e., is polymorphic, across genome copies and individuals. There are four possible nucleotide molecules: adenine, cytosine, guanine, and thymine. Nucleotides are abbreviated using the letter that beings the name of the molecule, so A, C, G, and T are the possible alleles of a SNP genotype.

1.1 Molecular Haplotyping Methods

Molecular methods exist to determine haplotypes by direct means using an individual's DNA, but these are of limited utility because they are time consuming, labor intensive, and error prone. The most prominent molecular haplotyping technique is allele-specific polymerase chain reaction (PCR). This technique requires somewhat less work than others, but it only gives information over extremely short genomic distances. It works by producing millions of copies of DNA segments from one of the homologous chromosome copies across a short region. The segments are then isolated using gel electrophoresis and sequenced to determine the haplotype for that chromosome copy over the short region of interest. A primer is a short segment of DNA that anneals to the target DNA at the location of interest and starts the copying process for PCR. In allele-specific PCR, the primer includes a SNP locus and contains the nucleotide that is complementary to (i.e., forms a base-pair with) one SNP allele. The aim of this design is for the primer to anneal only to the homologous chromosome copy that contains the specific complementary allele.

Unfortunately, the allele-specific primer can often mis-anneal to the alternate chromosome copy, and in that case, segments get copied from both homologous chromosomes. Because sequencing cannot differentiate between these two segments, mis-annealing of the allele-specific primer produces the same information as standard genotyping.

Besides obtaining segments from both chromosome copies, another error that can occur with allele-specific PCR is the construction of segments that do not exist *in vivo*. The PCR copying process forms DNA segments of varying lengths, and it often extends copying on short segments. Since both genome copies are substantially similar, these short segments can also mis-anneal, and the copying be extended from the alternate chromosome. In this case, the resulting "haplotype" segment contains portions of the haplotypes from both chromosome copies and does not not actually exist *in vivo*.

1.2 Computational Haplotyping Methods

The limitations of molecular-based haplotyping techniques have spurred the development of computational approaches in order to quickly and reliably infer genome-scale haplotypes. With the space of possible haplotypes being exponential in size, haplotyping algorithms must make assumptions based on genetic models to efficiently and accurately compute haplotype reconstructions.

Algorithms for inferring haplotypes can be separated into two broad classes. One class of haplotyping algorithms applies to unrelated individuals or to many family *trios* that contain data for a father, mother, and one child. Techniques of this class use probabilistic constraints governed by mathematical models of population dynamics. Algorithms of this class [23, 21] include PHASE [26], BEAGLE [2, 3], HAPLOTYPER [24], and HAP2 [19, 20]. The models these algorithms approximate are often insufficient to prevent switch errors—i.e., positions with incorrectly assigned haplotypes relative to the previous heterozygous locus [20, 26]—except across short genomic distances. Data from unrelated individuals have been used in many studies, notably in the International HapMap project [27], but such studies can only discover information about the results of meiosis (including the location of hotspots) averaged across thousands of generations and both genders.

The second class of haplotyping algorithms applies to individuals with known family relationships [8, 16, 1, 22, 14, 10, 11, 17, 18]. These algorithms infer haplotypes using the laws of Mendelian inheritance and the fact that allelic variants in close proximity to each other segregate together (i.e., genetic linkage). Haplotypes inferred from family-based data are accurate across a large genomic region. Depending on the family size, they will contain few or no switch errors. Additionally, these datasets and algorithms enable the identification of the probable sites of *de novo* meiotic recombinations and gene conversions (which appear as short double crossovers), and have been used to build genetic maps of recombination rates [13] and identify hotspots [4]. Considering *de novo* meiotic recombinations and gene conversions enables the study of differences in location and number of such events between individuals, including gender-based differences. Haplotypes for family-based datasets are also used to perform linkage analysis to study the genetic basis of disease within families.

1.3 Hapi

Hapi is a new dynamic programming algorithm that infers both minimum-recombinant and maximum likelihood haplotypes for nuclear families. Hapi has polynomial runtime in practice and performs substantially faster than existing algorithms, which have exponential runtime in general. Nuclear family derived genotypic data identifies parents and their children, but provides no information about relationships within a larger pedigree. Minimum-recombinant haplotypes assign family members' genotypes to homologs such that the number of recombinations that occur in the homologs the parents transmitted to the children is minimized. Maximum likelihood haplotypes utilize user-provided recombination frequencies between successive loci to calculate the most likely haplotype reconstruction.

Maximum likelihood haplotypes are often substantially similar or identical to minimum-recombinant haplotypes. Both approaches to haplotype estimation have strengths and weaknesses. Minimum-recombinant haplotyping may yield suboptimal results when the recombination frequencies between loci in some region varies widely. For example, because recombination frequencies are correlated with genomic distance, this can occur if the distance between some loci is much larger or smaller than others. Maximum-likelihood haplotyping reports only the most likely haplotype, a feature that can be misleading to a user when the difference in probability to alternate haplotypes is small. Typically this occurs when the number of recombinations across the alternate haplotypes are the same, and in such a case, minimum-recombinant haplotyping reports the ambiguities. It is worth noting that geneticists manually perform minimum-recombinant haplotype assignment when analyzing small datasets. Hapi enables this approach to be applied to the very large datasets currently produced by high-throughput SNP genotyping.

1.3.1 Summary of Hapi's Optimizations

Hapi is similar to existing haplotyping approaches that use HMMs and dynamic programming, but it employs a novel set of optimizations that dramatically improve its efficiency. These optimizations are based on properties we discovered about the haplotyping problem that facilitate the elimination of a large number of states from the analysis, and reduce the work required to transition between states at adjacent loci. A summary of the optimizations Hapi implements follows:

1. When a parent is homozygous at a locus, Hapi only builds states for that locus in which the homolog that that parent transmitted does not exhibit recombination relative to the previous locus. In connection with this, Hapi does not build states at loci where both parents are homozygous since recombination cannot be observed at these loci. This optimization is natural for minimum-recombinant haplotyping, but it requires special consideration in the context of maximum likelihood haplotypes as we discuss later (see Chapter 3).

2. At loci where Mendelian inheritance cannot unambiguously infer for a set of children which parent transmitted each allele, Hapi uses a novel, concise representation of the ambiguities instead of forming an exponential number of states for all the possibilities across all the children. It also avoids building any states that represent recombinations on both homologs for any ambiguous children and later evaluates whether such recombinations are consistent with nearby loci.

3. To transition between states at adjacent loci, Hapi considers a state at the previous locus as possibly transitioning to either two or four states at the next locus, depending on the genotypes and possible phase assignments of the parents at that locus. This optimization is actually a by-product of the first two optimizations mentioned above, but deserves separate consideration. Normally if two adjacent loci each have s states, there are s^2 possible state transitions (note that s may be an exponential number). [15] introduced a fast Fourier transform optimization that reduced the computational burden for transiton calculations to $O(s \cdot \log s)$, but Hapi's transition runtime is only $O(s)$, i.e., linear in the number of states at a locus.

4. Some states encode the same transmissions of homologs from the parents to the children and differ only in how the parents' alleles are assigned to homologs. These states are equivalent downstream of the current locus and Hapi only retains the one with minimum recombinations or maximum likelihood. Kruglyak et al. originally discovered a more general form of this optimization that applies to all founders in a pedigree [14]. Hapi applies this optimization to parents in a nuclear family.

5. The above optimization that removes states with equivalent homolog transmissions from parents to children is most effective when none of the children are missing genotype data. We devised a mechanism for comparing nearly equivalent states in the presence of children with missing data that often enables the detection and elimination of suboptimal states.

6. At each locus, Hapi only considers states that are consistent with Mendel's laws for the genotypes of the individuals and spends no time processing any inconsistent states. Other algorithms also employ similar optimizations that help reduce the number of states they examine [22, 1, 11].

We present a detailed discussion of each of these optimizations in Chapters 2 and 3.

1.4 Other Family-Based Haplotyping Algorithms

Several existing programs for haplotyping related individuals are based on the Lander-Green algorithm [16], including Merlin [1], GENEHUNTER [22, 14], and Allegro [10, 11]. The basic approach of the Lander-Green algorithm uses hidden Markov models (HMMs) to obtain a probability distribution of haplotype assignments for individuals in a pedigree. A user can either sample a haplotype from this distribution, or, more commonly, obtain the maximum likelihood haplotype assignment. The state space for these HMMs is composed of inheritance vectors at each locus that are bit strings encoding which chromosome homolog a parent transmitted for each child in the pedigree at that locus. This state space is inherently exponential, with 2^{2n} possible values, where n is the number of non-founders or individuals with at least one parent in the pedigree.

Although Merlin, GENEHUNTER, and Allegro all employ techniques to reduce space and time requirements of this basic algorithm, all are relatively inefficient; in general, each requires exponential time in the number of non-founders in the pedigree. One technique that all these algorithms employ is to avoid representing inheritance vectors that are inconsistent with Mendelian inheritance. In addition, Merlin [1] uses sparse gene flow trees that avoid redundant representations for states with identical likelihoods or a probability of zero. Allegro [11] uses multi-terminal binary decision diagrams (MTBDDs) [7], which are more general than sparse gene flow trees. MTBDDs are at least as sparse as Merlin's sparse gene flow trees, and depending on how they are constructed, can be smaller. The optimized representations that Merlin and Allegro utilize are effective in reducing the state representations at a single locus. However, transition probabilities will, in general, differ for most or all possible transitions between states at adjacent loci. Because of this, the algorithms must represent most or all of the 2^{2n} states in order to perform multipoint analyses, including haplotyping.

Superlink [6] is another maximum likelihood haplotyping algorithm that uses Bayesian networks. While Superlink employs several optimizations to improve its efficiency, it performed slower than Merlin and Allegro in our experiments.

To compare Hapi's runtime performance with existing work, we ran Merlin, Allegro, Superlink and Hapi on a dataset containing 103 nuclear families. (We excluded GENEHUNTER from these comparisons since Merlin has faster runtime [1].) Hapi performed 3.8–320 times faster than Merlin, 6.4–2460 times faster than Allegro. Superlink was unable to haplotype the entire dataset, so we ran it on a slightly smaller version. Hapi ran 17–448 times faster than Superlink for this modified dataset (see Chapter 4). We also compared Hapi to PedPhase 2.0 [18], a system for computing minimum-recombinant haplotypes. Whereas Hapi finished in 4.732 seconds, PedPhase did not complete analyzing even one chromosome in over six hours' time. These results demonstrate the efficacy of Hapi's optimizations in the context of real genotype data.

Existing algorithms have limits in the size and number of families they can haplotype. With Hapi, the efficient haplotyping of very large numbers of families as well as families with large number of individuals is now possible. Because of the relative ease of gathering genotypes for nuclear families, we expect that the number of nuclear families within datasets will continue to grow and that Hapi will provide the opportunity to haplotype this large quantity of data.

Besides haplotyping nuclear families, this thesis presents an algorithm for extending the techniques in Hapi in order to haplotype multi-generational pedigrees (see Chapter 7). Current haplotyping algorithms cannot scale beyond pedigrees containing more than roughly 20 non-founders. This extension to Hapi promises to make possible the haplotyping of moderate to large multi-generational pedigrees that current techniques cannot haplotype.

1.5 Contributions

This thesis makes four primary contributions:

1. Hapi, a new algorithm for performing minimum-recombinant and maximum likelihood haplotyping for nuclear families that includes several novel optimizations.

2. An implementation of the Hapi algorithm and an evaluation of Hapi compared to several state-of-the-art techniques. This comparison demonstrates Hapi's efficiency in haplotyping real human genotype data for set of nuclear families: Hapi performs orders of magnitude faster than other techniques.

3. An algorithm for extending Hapi to haplotype either particular loci or entire nuclear families that are missing genotypes for one or both parents.

4. An algorithm for extending Hapi to haplotype multi-generational pedigrees. This novel approach provides the possibility of haplotyping moderate or large pedigree datasets that no existing algorithm can currently handle.

1.6 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents our minimum-recombinant haplotyping algorithm and includes a detailed discussion of the optimizations implemented in Hapi. Chapter 3 describes the mechanism for modifying this minimum recombinant algorithm in order to efficiently perform maximum likelihood haplotyping. Chapter 4 provides experimental results comparing Hapi to existing state-of-the-art haplotyping algorithms and gives a run-time complexity analysis of the Hapi algorithm. Chapter 5 explains how Hapi could be used to perform linkage analysis and genotype or sequence imputation. Chapter 6 describes how to extend Hapi to handle loci and nuclear families that do not include genotype data for one or both parents. Chapter 7 presents an algorithm for extending Hapi to efficiently haplotype multi-generational pedigrees, and Chapter 8 gives conclusions.

Chapter 2

Minimum-Recombinant Haplotyping

Minimum-recombinant haplotype assignments apply to family-based datasets and are those with the minimum number of recombinations in the chromosomes that parents transmitted to their children. Hapi infers haplotypes for nuclear families, but the minimum-recombinant definition also applies to pedigrees: the count is across all chromosome transmissions for every parent–child relationship in the pedigree. This minimum-recombinant count is summed across all children and over the entire chromosome length. Hapi seeks, therefore, to find a haplotype solution that is *globally* minimal across the chromosome length rather than *locally* minimal between successive pairs of loci. Thus, a solution may contain a locus that has an alternate assignment of individuals' alleles to homologous chromosomes that yields fewer recombinations from the previous locus (i.e., locally), but not over the entire chromosome length (i.e., globally). An example of such a locus from real data for a family of human subjects is shown in Figure 2-6, which we discuss in Section 2.9.

In order to find a minimum-recombinant haplotype solution, Hapi explores all possible haplotype assignments by an approach that uses dynamic programming and applies a novel set of optimizations.

2.1 Inheritance Vectors

As in related algorithms, Hapi uses inheritance vectors, which are stored as bit vectors, to encode which chromosome homolog the parents transmitted to each child at a locus. Given an assignment of parents' alleles to homologs at a locus—i.e., a haplotype assignment, also known as a *phase* assignment—inheritance vectors identify which homologs the parents transmitted to the children. Inheritance vectors imply the children's haplotypes since they indicate which homolog, and therefore which allele, the parent transmitted.

An inheritance vector for a single locus contains $2c$ bits, where c is the number of children in the nuclear family—one bit for the result of each meiosis. Since parents have two homologous chromosomes, it suffices to identify the homologs with a single

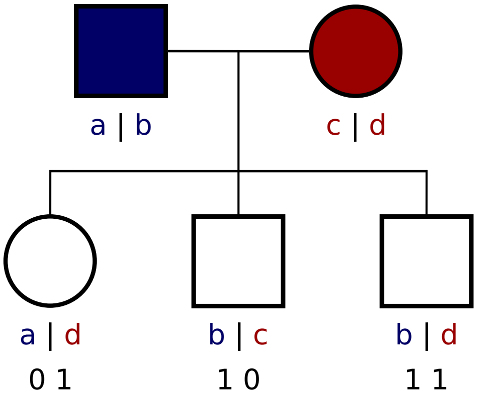


Figure 2-1: A nuclear family with phased alleles for a single locus and the corresponding inheritance vector values for each child. The inheritance vector for this locus is the composite of the individual child inheritance bits: `011011`.

bit, either 0 or 1, that indicates which homolog a parent transmitted at a locus.

Figure 2-1 shows an example three-child family with phase assignments for each person at a single locus and the corresponding inheritance vector values for the children. Because the first child received allele *a* from the father, i.e., the allele on the father's first homolog, the corresponding inheritance vector bit is `0`. Similarly, this child received allele *d* from the mother, or the allele from the mother's second homolog, so the resulting inheritance vector bit is `1`. Overall, the inheritance vector for these three children is `011011`—the concatenation of the inheritance bits for each of the children.

2.2 Minimum-Recombinant Haplotyping Equation

To haplotype a chromosome, Hapi considers each locus successively in the order they reside physically, moving from one end of the chromosome to the other. Starting at one end of the chromosome is not strictly necessary. The algorithm produces equivalent results for any starting point, and can process the loci sequentially in either direction, as long as it processes adjacent loci in succession. Proceeding in order in one direction along a chromosome is necessary to enable the most fine-grained comparison of inheritance vector values (i.e., counting of recombinants) between loci and thereby produce optimal minimum-recombinant (or maximum likelihood) haplotypes.

A dynamic programming equation for calculating minimum-recombinant haplotypes is given below. The function $R(l, \vec{v})$ calculates the minimal number of recombinations necessary to reach inheritance vector $\vec{v}$ at locus l:

$$R(l, \vec{v}) = \min_{\vec{w}} \{R(l-1, \vec{w}) + H(\vec{w}, \vec{v})\}. \tag{2.1}$$

Here, $R(l-1, \vec{w})$ is the minimum number of recombinations necessary to reach an inheritance vector $\vec{w}$ at the previous locus $l-1$. $H(\vec{w}, \vec{v})$ is the number of recombinations between vectors $\vec{w}$ and $\vec{v}$, which is equal to the number of bits that differ between them, i.e., the hamming distance. The initial number of recombinations at locus $l = 0$ is defined natural as $R(l = 0, \vec{v}) = 0$.

An efficient implementation of the hamming distance function uses the exclusive-or operator on the two inheritance vectors, which Hapi stores as unsigned integers. For every bit position where the two operands differ, the exclusive-or result contains a `1` bit in that position; if the two operand bits are the same, the corresponding bit is `0`. Counting the number of bits in the result of the exclusive-or operation gives the number of recombinations between the inheritance vectors.

A naive implementation of the dynamic programming recurrence given in equation (2.1) would initialize all 2^{2c} possible inheritance vectors at locus $l = 0$ and would model most or all of these vectors at successive loci. Hapi functions differently: the initial locus has only one inheritance vector, and successive loci model a very small number of inheritance vectors.

2.3 Hapi States

Hapi uses what we term a locus *state* to store the information computed in the dynamic programming equation. A locus state stores:

1. An inheritance vector.

2. The assignment of the heterozygous parent's or parents' genotype alleles to homologs that is consistent with this inheritance vector.

3. The minimal number of recombinations necessary to reach this state/inheritance vector value.

4. A pointer to the state or states at the previous locus that yields this minimal number of recombinations.

5. A bit vector encoding which children have ambiguous inheritance values (necessary for some kinds of loci as we describe in Section 2.4.2).

Because the parents' allele to homolog assignments imply part or all of the inheritance vector values, there is only one consistent parent assignment for each inheritance vector. Note that it is possible for multiple states at the previous locus to yield the minimum number of recombinations at the current locus, i.e., there are sometimes ambiguities. We discuss this issue in Section 2.7.

2.3.1 Back Tracing

After evaluating equation (2.1) by building the necessary states for all loci, it is straight forward to deduce haplotypes. Hapi does this by performing the assignments of alleles to homologs as dictated by the minimum-recombinant state at the final locus and then back tracing to states at previous loci. Assigning alleles to homologs is simple for the parents since the state explicitly stores these assignments. For

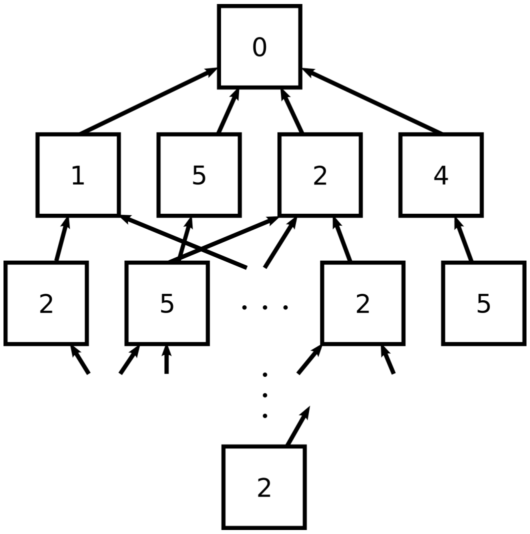


Figure 2-2: A pictorial representation showing the relationship between states at different loci. Each row of boxes correspond to a locus; boxes represent a state and indicate the numbers of recombinations the state incurs; arrows point to previous state(s). Once the system deduces a single state at some locus—shown here as the bottom box—it back traces by traversing the pointers and assigns the haplotype values from the states it encounters. The numbers are not from real data.

the children, the inheritance vectors encode a bit for each parent–child relationship that designates which homolog the parent transmitted to the child at that locus. It suffices to designate one homolog for each child as transmitted by each of the parents and assign the allele that a given parent transmitted to its respective homolog. After performing these allele assignments for the family at the final locus using its minimum-recombinant state, Hapi back traces by following the pointer from this final state to the state at the previous locus and repeats the process again.

Rather than waiting until the final locus to make these assignments and perform back tracing, Hapi does this work whenever a locus yields only one state (which happens frequently). The one state at that locus and those leading to it at previous loci are guaranteed to have minimum recombinations. Performing this process before the final locus allows the system to reclaim the memory used to store states.

We give an illustrative example of what the graph of states generated by our algorithm may look like in Figure 2-2. In this graph, boxes represent states, and each row of boxes corresponds to the states for a single locus. The number in each box represents the minimal number of recombinations necessary to reach that state. The first locus (top-most box) has only one box/state with an initial value of 0 recombinations. At the second locus, there are four states that have between 1 and 5 recombinations. Note that at the third locus, the second state has pointers to two different states at the previous locus. The final locus has only one state. Once the system determines this final state, it performs back tracing along pointers to previous states, and uses the haplotype values stored in the encountered states to make the allele assignments as we just described.

2.3.2 Building States

Hapi builds states for a locus based on the states at the previous locus and the genotypes of the individuals at the locus being considered. Utilizing states from the previous locus is necessary for two of Hapi's optimizations: not introducing recombinations at a locus where a parent is homozygous (see Section 2.4.1), and the ambiguous inheritance vector values optimization (see Section 2.4.2). The system also uses previous locus states at loci where children are missing data (see Section 2.8). The initial state for a chromosome cannot depend on previous locus states and is therefore built differently. We describe the process for building an initial state in Section 2.6.

Hapi builds states for every possible phase assignment for the heterozygous parent(s). For each such assignment, Hapi determines which alleles the heterozygous parent(s) transmitted to each child and assigns the corresponding values in the inheritance vector accordingly. The algorithm assigns the inheritance values for a homozygous parent as we describe in Section 2.4.1.

When both parents have the same heterozygous genotype, a heterozygous child will have the same genotype as both its parents and could have received either allele from either parent. The corresponding inheritance vector values for each such child is ambiguous, and in this case, Hapi uses its optimization of a novel representation to encode the possible inheritance vectors in a single state (see Section 2.4.2).

We next describe each of the optimizations Hapi implements. The goal of each optimization is to reduce the number of states Hapi must build and retain. A side effect of these optimizations is to reduce the number of state of possible state transitions between loci, as the next section explains.

Later, in Section 2.5, we present a classification of loci into four locus types that are based on the parents' genotypes at a locus. Different optimizations apply to each locus type, and that section describes and gives examples of how these optimizations apply.

2.4 Hapi's Optimizations

Hapi implements six optimizations that allow it to very efficiently infer minimum-recombinant haplotypes. Hapi also utilizes these optimizations to calculate maximum likelihood haplotypes, as we describe in Chapter 3.

This section provides details about five of the optimizations Hapi implements. The last optimizations applies at loci where one or more children are missing data, and we discuss it in Section 2.8.

2.4.1 Non-Recombinant States for Homozygous Parents

When one or both of the parents at a locus are homozygous, the alleles are identical, and therefore, which homolog the homozygous parent(s) transmitted is ambiguous. A naive implementation of the Lander-Green algorithm builds states corresponding to all possible homolog transmissions for the homozygous parent. This yields 2^c

inheritance vector values for each homozygous parent, which produces at least that many states at the locus in question. Consider a locus with one heterozygous parent and one homozygous parent. If s is the number of possible inheritance vector values corresponding to the heterozygous parent, then there are $s \cdot 2^c$ possible states.

Instead of building and tracking this exponential number of states, Hapi copies the inheritance vector values corresponding to the homozygous parent from the states at the previous locus. There are typically a small number of unique inheritance vector values for the homozygous parent in the states at the previous locus. If there are t unique values for the homozygous parent's inheritance bits at the previous locus, there will be $s \cdot t$ states. Note that while t can be large, or even an exponential number, other optimizations serve to limit the number of states in general. Our experimental results demonstrate that the number of states is low in practice.

This approach of copying inheritance vector values for the homozygous parent assumes a lack of recombination for this case when the results of meiosis cannot be observed; this will always yield minimal recombinations. The next locus that is heterozygous for the parent in question indicates if a recombination has occurred. The exact location of any such recombination is unknown—the maximum likelihood approach makes use of recombination frequencies to estimate the location—but must be between that later locus and the most recent upstream locus heterozygous for the parent under consideration.

For loci where both parents are homozygous, all 2^{2c} possible inheritance vectors are consistent with the genotypes since the homologs in each parent contain the same allele at this locus. Rather than tracking all possible states at loci where both parents are homozygous, and rather than copying exactly all the states from the previous locus, Hapi simply skips these loci. Subsequent loci utilize the states located at the most recent locus for which states exist; this locus will be the most recent one with at least one heterozygous parent.

Figure 2-4 shows states for a locus in which one parent is homozygous and the other parent is heterozygous. The inheritance vector values corresponding to the homozygous parent p_1 are shown as the second element in each of the ordered pairs in the rows labeled $\vec{v}$. The inheritance vector values for the homozygous are parent in the two states a and b are the same as those in the previous state since, as we have described, Hapi copies these values. Without some form of optimization for this case, the two states shown would instead correspond to $2 \cdot 2^c$ states.

Other algorithms, including Merlin [1] and Allegro [11], have techniques to reduce the number of states they represent in the presence of uninformative meioses. Merlin uses sparse binary trees to encode inheritance vector probabilities and has an optimization that applies to homozygous parents. The bit value of an inheritance vector determines which edge to traverse through the tree. A full tree would contain $2^{2c+1} - 1$ nodes, but typically there are much fewer than 2^{2c} nodes. For a homozygous parent, either bit value yields the same probability and Merlin places an internal "symmetric node" to indicate that the tree stored for the alternate bit yields identical results. (Note that Merlin uses the binary trees to determine the probability of inheritance vectors at the current locus. It calculates state transition probabilities separately, performing calculations across many more states than Hapi does. See Section 2.4.3

for how Hapi transitions between states.) This optimization is effective, but Merlin must still build a large number of states. At loci where both parents are homozygous, the system builds $2c$ nodes; if one parent is homozygous, it builds roughly $4c$ nodes.

2.4.2 Ambiguous Inheritance Vector Values

At loci for which both parents are heterozygous with the same genotype (which we later term “partly informative”), heterozygous children have the same genotype as their parents. As a result, these children are *a priori* ambiguous as to which parent transmitted each of their alleles: either parent could have transmitted either allele.

Existing algorithms build states corresponding to all possible inheritance vector values for these ambiguous children. For a given assignment of the parents’ alleles to homologs, each heterozygous child has two possible inheritance vector values. For h heterozygous children, there are 2^h possible inheritance vectors for each of the four possible assignments of parents’ alleles to homologs. Therefore, there are $4 \cdot 2^h$ inheritance vectors/states consistent with individuals’ genotypes at these loci.

Instead of building this exponential number of states, Hapi again uses the states at the previous locus to reduce the number of states it must build. The system maps each previous state to four states corresponding to each assignment of parents’ alleles to homologs. Note that multiple previous states can map to the same state, so the number of states usually does not quadruple. Also note that homozygous children have only one inheritance vector value that is consistent with a given assignment of parents’ alleles, so they do not increase the number of necessary states.

Hapi assigns heterozygous children’s inheritance vector values based on the inheritance values in the previous state. These children have two consistent inheritance vector values for a given assignment of parents’ alleles to homologs, and these two values are opposite each other. If the inheritance value in the previous state is equivalent to one of these two values, Hapi uses the value equivalent to the previous state in the state being built. The other inheritance value results in two crossovers for the child, one from each parent. Such an event is extremely unlikely, yet if it were to take place, downstream loci that are fully informative would reveal its occurrence. In that rare case, Hapi will mark the partly informative locus as ambiguous during back tracing, since it is impossible to know whether these two recombinations took place at the earlier partly informative locus or at the later fully informative locus. (Maximum likelihood haplotyping determines the location of the recombinations based on recombination frequencies.)

In the case that the inheritance value in the previous state is not equal to one of the two ambiguous inheritance values, the previous inheritance value must differ from these two values in exactly one bit. For example, if the previous value is $\langle 0, 0\rangle$ and is not equal to either of the values at the current locus, they must be $\langle 0, 1\rangle$ and $\langle 1, 0\rangle$. (Otherwise one of the values would be $\langle 1, 1\rangle$, and the opposite value of $\langle 0, 0\rangle$ is equal to the previous value.) The differences between the two consistent values and the previous one represent a recombination in one or the other parent. Which parent recombined is ambiguous at this locus and can only be determined at later loci.

Rather than creating separate states for these two inheritance values—which would yield an exponential number of states across multiple children—Hapi instead marks the child as having ambiguous inheritance. A child's inheritance being marked as ambiguous means that its inheritance vector value can be inverted without inducing additional recombinations—both possibilities result in the exactly one recombination. The choice of which of the two inheritance values to store in the state is arbitrary, and Hapi indicates that a child is ambiguous using another bit vector. For our explanation here, we designate ambiguous values with the ? symbol. One can view an ambiguous inheritance value as a set of values, so $\langle 0,0\rangle? = \langle 1,1\rangle? = \{\langle 0,0\rangle, \langle 1,1\rangle\}$. For the earlier example with a previous inheritance value of $\langle 0,0\rangle$, the resulting inheritance value would be $\langle 0,1\rangle?$. The use of these ambiguous values effectively merges the exponential number of states that would otherwise result. Merging the states in this way suffices because (1) Hapi can later resolve which of the unambiguous inheritance vectors is optimal, and (2) the number of recombinations remains the same regardless of which unambiguous inheritance vector ultimately results.

If the previous inheritance value is itself ambiguous, the resulting value must also be ambiguous. If there is a recombination, the resulting value is unequal to the previous value, such as with $\langle 0,0\rangle?$ and $\langle 0,1\rangle?$.

Hapi resolves ambiguous inheritance values for a state during the back tracing process discussed earlier (see Section 2.3.1). While back tracing, if the system encounters a state that has one or more ambiguous inheritance values, it compares these values to the corresponding values at the next—already resolved—locus. If the unambiguous form of this value (i.e., that without the ? symbol) or its opposite is equal to the inheritance value at the next locus, the system assigns the equivalent value at the current locus. If neither is equal, recombinations occur on either side of this locus and the inheritance value is truly ambiguous. In this rare case, Hapi's output reports the child's haplotype at this locus as ambiguous.

This optimization significantly improves Hapi's efficiency. Removing this optimization would cause the number of states to grow unwieldy whenever Hapi encountered a locus that has heterozygous parents with the same genotype. Even with all the other optimizations in place, the increase in the number of states would propagate to subsequent loci that have one parent that is heterozygous and the other homozygous. The optimization that applies to such loci propagates inheritance vector values from the previous state and is effective when the number of such states is small.

GENEHUNTER [22] identifies some constraints on inheritance vector values that relate to those we describe above. However, merging an exponential number of inheritance values into a single state through the use of ambiguous inheritance values is novel and unique to Hapi. Our ambiguous inheritance values optimization reduces the state space much more significantly than do the ideas described in GENEHUNTER. As well, the authors of GENEHUNTER chose not to include the optimization relating to this sort of scenario in the program itself because a non-trivial number of operations are necessary to check for constraint violations.

2.4.3 State Transitions Between Loci

The optimizations in the previous two subsections (Sections 2.4.1 and 2.4.2) describe how Hapi uses the states at the previous locus to build states at the current locus.

Besides reducing the number of states at a locus, these optimizations also reduce the number of possible state transitions between loci. In general, any state at a previous locus can transition to any state at the next locus. However, because Hapi does not consider state transitions that include recombinations from a parent that is homozygous, and because it uses ambiguous inheritance values, the number of possible state transitions is limited. The state transition optimization that this subsection describes actually comes as a by-product of the two optimizations we have already outlined. Yet the effects of these optimizations on the complexity of state transition calculations merit a separate discussion.

At each locus, Hapi considers transitions from the states at the previous locus to either two or four states. If only one parent is heterozygous at the locus, each state at the previous locus can transition to only two states at the current locus. These two states correspond to the two possible phase assignments for the heterozygous parent. A particular phase assignment for the heterozygous parent uniquely defines the inheritance vector bits that that parent transmits. The system copies the other inheritance vector bits from the previous state, which can only transition to states with those inheritance bits (see Section 2.4.1). Thus, when one parent is heterozygous, each locus at the previous state can only transition to two states at the current locus.

If both parents are heterozygous at a locus, then the parents have four possible phase assignments, and each state at the previous locus can transition to four states at the next locus. The ambiguous inheritance vector optimization makes this possible. Heterozygous children at loci in which both parents have the same heterozygous genotype would otherwise produce an exponential number of states. Instead, for a given phase assignment for the parents, a state at previous locus uniquely determines the inheritance vector it can transition to (see Section 2.4.2). If the parents are heterozygous with differing genotypes, the children's genotypes at the locus unambiguously determine the complete inheritance vector that corresponds each parent phase assignment. Thus, exactly four inheritance vectors are possible, and each previous state can transition to these four states.

The efficiency gains of our approach are significant. Without this optimization, haplotyping algorithms must consider all possible state transitions between loci. If two adjacent loci each have n states, other algorithms compute transition probabilities corresponding to all n^2 state transitions. Use of a fast Fourier transformation reduces the computational burden of these optimizations from a quadratic $O(n^2)$ to $O(n \cdot \log n)$. With Hapi's optimizations there are only $2n$ or $4n$ possible transitions, so the computational burden is linear, $O(n)$. The speed of computing state transitions—in addition to and in connection with tracking of very few states at each locus—enable Hapi to perform haplotyping calculations very efficiently.

2.4.4 Equivalent States

At many loci, it is possible to unambiguously deduce which allele each heterozygous parent transmitted to each child. In that case, the bits that correspond to transmissions from this parent can take on exactly two values depending on the parent's phase assignment. The inheritance bits in these two values are opposite each other, since the parent transmits the same allele in each case, but the alleles reside on opposite homologs for the opposite phase assignments. The locus in Figure 2-4 illustrates these ideas. For this locus, it is easy to deduce which alleles the heterozygous parent transmitted to each child. As well, the two states have opposite inheritance values corresponding to this parent, consistent with their opposite phase assignments.

Two inheritance vectors with opposite bits corresponding to one parent and equivalent bits for the other parent are equivalent in terms of the number and locations of recombinations at downstream loci. Hapi uses inheritance vectors to detect recombinations. A recombination occurs when the homolog a parent transmitted to a child differs between two loci. Because the parent's inheritance values in these states are exactly opposite each other, each of these inheritance vectors encodes the same set of children as receiving a given homolog. The two states merely use opposite labels for the homologs as implied by the parent's opposite phase assignments. Choosing one of the states instead of the other results in all downstream loci having opposite phase assignments for the parent, consistent with the chosen phase assignment in the upstream state. The number and location of downstream recombinations are the same regardless of which state the system chooses at this locus because the sets of children that share a common homolog are the same between states.

In general, any states with opposite inheritance values for one parent and either equivalent or opposite inheritance values for the other parent are equivalent. This means that, if both parents are heterozygous, there are four states that are equivalent to each other and only the state with the fewest recombinations must be retained. Hapi detects these states at loci and eliminates any suboptimal states as needed.

Kruglyak et al. [14] first discovered a more general form of this optimization, finding that equivalent states exist for all founders in a pedigree. A founder is an individual with no parents in the pedigree. For each founder, the number of inheritance vectors is decreased by a factor of 2. So, whereas there are 2^n possible inheritance vectors in a pedigree, where n is the number of non-founders, this optimization reduces the state space to 2^{2n-f} inheritance vector, where f is the number of founders. For a nuclear family, $f = 2$, so this optimization reduces the state space by a factor of 4.

2.4.5 States Consistent with Mendel's Laws

Although there are 2^{2c} possible inheritance vectors for every locus, the genotypes of the individuals at a locus often make many of these inheritance vectors inconsistent with the Mendelian laws of inheritance. For example, a parent that is heterozygous with genotype a/b cannot transmit its b allele to a child that is homozygous with genotype a/a. Hapi builds states based explicitly on the genotypes at each locus and spends no time processing any inheritance vectors that are inconsistent with the genotypes at the locus.

Merlin [1], GENEHUNTER [22], and Allegro [11] all contain optimizations that reduce the number of nodes/states each represents on the basis of inconsistent inheritance vectors. In Merlin's sparse binary trees, when a given branch of the tree has an impossible inheritance vector, the algorithm uses a premature leaf node with 0 probability to reduce the number of nodes. Even so, the trees will contain many nodes to reach various terminal branches for level of the tree (which corresponds to each meiosis). GENEHUNTER uses bit masks that encode which inheritance vector bits must be set to a certain value in order to be consistent with Mendelian inheritance. It thereby forgoes representing any inheritance vectors it determines to be inconsistent with the genotypes at a locus.

The various packages' optimizations are roughly equivalent, though Hapi functions differently in that it does not spend any computation on incompatible inheritance vectors—other algorithms spend some small amount of time ruling them out. Each program's optimizations provide a significant reduction in the state space that each considers. Though the storage and computational approaches differ, deducing the results of meiosis for one parent removes roughly half the inheritance vector values from the analysis.

2.5 Locus Types

Hapi's optimizations apply in different contexts. In particular, we have identified four types of loci with different parents' genotypes for which different technical issues arise and different optimizations apply. Figure 2-3 summarizes these locus types, listing the number of states that result at each type if there are n states at the previous locus. The figure also includes the average number of states that occur at relevant locus types for the dataset we evaluate in Chapter 4. Further details about each type and the number of necessary states follow.

The figure lists names we use to refer to each locus type. In the genetics community, a parent's locus is said to be *informative* if it is heterozygous. Such loci are informative for meiosis—they can be used to determine which chromosome homolog a parent transmitted and thus whether a recombination occurred relative to some previous locus. By their nature, homozygous loci are uninformative because their two alleles are identical. One cannot distinguish which of the two alleles or chromosome homologs a homozygous parent transmitted to its child.

We now describe each of the locus types in turn and provide more details about Hapi's implementation.

2.5.1 Fully Informative for Both Parents Loci

At a fully informative for both parent locus, Hapi keeps only one state regardless of the number of states at the previous locus. Both parents are heterozygous but have differing genotypes at this type of locus. In this case, it is simple to deduce which allele each parent transmitted to each child; thus, for a particular phase assignment for both parents, there is only one possible inheritance vector. With two possible phase assignments for each of the two parents, there are exactly four inheritance vectors/states that are consistent with Mendel's laws at this type of locus.

The four states Hapi builds have either opposite or identical inheritance vector values corresponding to each parent. As such, the equivalent states optimization ap-

Locus Type	Parent p	Parent q	Number of States	
			if n previous states	Avg
Fully Informative for Both Parents	a/b	a/c or c/d	1	N/A
Fully informative for One Parent	a/b	a/a or c/c	After informative for parent q: 1 Previous states unambiguous: $\leq n$ Previous states ambiguous: $\leq 2n$	1.81
Partly Informative	a/b	a/b	$\leq 4n$	6.04
Uninformative	a/a	a/a or b/b	0	N/A

Figure 2-3: The four types of loci our algorithm handles separately with the names we use to refer to them. The figure lists the number of states that Hapi produces for each type if there are n states at the previous locus, and gives the average number of states produced for haplotyping the dataset we evaluated in Chapter 4. We include further details in the sections that discuss each type. Note that either parent may have the genotypes listed for parents p and q.

plies at this locus type, and all four states are equivalent. Hapi therefore eliminates all but one state, retaining only the stat that has the minimum number of recombinations. With only one resulting state, Hapi back traces and reclaims the memory associated with states at upstream loci.

Note that though this locus type is advantageous, most SNPs are bi-allelic, and therefore this locus type will not occur in SNP genotype datasets. However, as we will see in the next section, two successive loci that are fully informative for each of the parents has the same effect as a fully informative for both parents locus.

2.5.2 Fully Informative for One Parent Loci

The number of states that Hapi retains at a fully informative for one parent locus depends on the previous locus type. There are three possibilities, and we describe each in this section. In brief, let n be the number of states at the previous locus. In general, if the previous locus states do not have any ambiguous inheritance values, Hapi produces n or fewer states at the current locus. If the previous locus is fully informative for the parent that is homozygous at the current locus, Hapi retains only one state at the current locus. Such a case is analogous to encountering one locus that is fully informative for both parents. Finally, if the previous locus states have ambiguous inheritance values, the number of states Hapi retains for this locus will be no more than $2n$.

To build states at this type of locus, Hapi first deduces the inheritance vector values corresponding to the heterozygous parent according to its two possible phase assignments. This defines two possibilities for half of the inheritance vector values, but leaves undefined the half corresponding to the homozygous parent. In Section 2.4.1, we described an optimization that copies the inheritance values for the homozygous parent from the states at the previous locus. This is the next step in processing this

		Parents		Children					# Rec
		p_0	p_1	c_0	c_1	c_2	c_3	c_4	
Prev	$\vec{v}$			$\langle 0, 1\rangle$	$\langle 1, 1\rangle$	$\langle 1, 1\rangle$	$\langle 0, 0\rangle$	$\langle 1, 1\rangle$	0
State	*hap*	$\langle \mathrm{a, g}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{g, a}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{a, a}\rangle$	4
a	$\vec{v}$			$\langle \mathbf{1}, 1\rangle$	$\langle \mathbf{0}, 1\rangle$	$\langle \mathbf{0}, 1\rangle$	$\langle 0, 0\rangle$	$\langle \mathbf{0}, 1\rangle$	
State	*hap*	$\langle \mathrm{g, a}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{g, a}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{a, a}\rangle$	$\langle \mathrm{a, a}\rangle$	1
b	$\vec{v}$			$\langle 0, 1\rangle$	$\langle 1, 1\rangle$	$\langle 1, 1\rangle$	$\langle \mathbf{1}, 0\rangle$	$\langle 1, 1\rangle$	

Figure 2-4: An example of a fully informative for one parent locus showing one state at the previous locus and the two states Hapi builds based on this previous state. This example is from the real dataset discussed in Chapter 4. The rows labeled $\vec{v}$ show the states' inheritance vectors and the rows labeled *hap* give haplotype assignments of the alleles. Hapi copies the inheritance vector values corresponding to the homozygous parent from the previous state to states *a* and *b*. Recombinations result from differing inheritance vector values from the previous state; these differences appear in bold and the states' total number of recombinations appear in the right-most column. Note that the heterozygous parent's inheritance vector values in the two states are exactly opposite each other and are therefore equivalently labeled.

locus type; the algorithm builds complete states at the current locus by copying the values for the homozygous parent. All pointers to previous states are to states with identical inheritance values for the homozygous parent. This ensures that the states never incur any recombinations for this uninformative parent. These steps result in $2n$ or fewer states—fewer if multiple previous states have the same inheritance values for the homozygous parent.

After building all states using the inheritance values from states at the previous locus, Hapi applies the equivalent states optimization (see Section 2.4.4). Each previous state can transition to two states at the current locus with equivalent inheritance vector values since the bits corresponding to the homozygous parent are the identical, and those corresponding to the heterozygous parent are opposite. Hapi therefore eliminates half the states that get built, retaining the state with fewer recombinations for each equivalent pair of states.

If the previous locus was heterozygous for the parent that is currently homozygous, only two inheritance values occur for that parent at the previous locus, and these are opposite each other. Therefore, in this case, Hapi builds four equivalent states, and it retains only the one with the fewest recombinations.

Note that Hapi utilizes three of its four optimizations—all but the ambiguous inheritance value optimization—at this type of locus. This includes the two optimizations mentioned above, as well as not building any states that are inconsistent with Mendelian inheritance.

Figure 2-4 gives an example from real data of a fully informative locus with one state at the previous locus. To make the inheritance vectors more readable, we display them using ordered pairs for each child. It can be seen in this example that the system copies the inheritance vector values for homozygous parent from the previous locus

		p_0	p_1	c_0	c_1	c_2	c_3	c_4	# Rec
Prev	$\vec{v}$			$\langle 0, 1\rangle$	$\langle 1, 1\rangle$?	$\langle 1, 1\rangle$	$\langle 0, 0\rangle$?	$\langle 1, 1\rangle$	0
State a	*hap* $\vec{v}$	⟨a, g⟩	⟨a, a⟩	⟨g, a⟩ $\langle \mathbf{1}, 1\rangle$	⟨a, a⟩ $\langle \mathbf{0}, \underline{0}\rangle$	⟨a, a⟩ $\langle \mathbf{0}, 1\rangle$	⟨a, a⟩ $\langle 0, \underline{0}\rangle$	⟨a, a⟩ $\langle \mathbf{0}, 1\rangle$	4
State b	*hap* $\vec{v}$	⟨g, a⟩	⟨a, a⟩	⟨g, a⟩ $\langle 0, 1\rangle$	⟨a, a⟩ $\langle 1, \underline{1}\rangle$	⟨a, a⟩ $\langle 1, 1\rangle$	⟨a, a⟩ $\langle \mathbf{1}, \underline{1}\rangle$	⟨a, a⟩ $\langle 1, 1\rangle$	1

Figure 2-5: An example, modified from Figure 2-4 and not from real data, showing a state with ambiguous inheritance values (marked by ?) at the previous locus, and the two states Hapi builds based on it. For unambiguous children's inheritance vector values, the system copies the bits corresponding to the homozygous parent from the previous state. For ambiguous children, two opposite inheritance values are valid for the previous state, and the system uses the homozygous parent bit from the inheritance value that matches the heterozygous parent's bit in the state being built. Both of the two inheritance values are necessarily represented, one in each of the resulting states. As the underlined values show, the inheritance values for the homozygous parent differ across the two outputs. As such, the states are not equivalent, and Hapi cannot eliminate either. Bold values indicate recombinations.

state. As well, the inheritance vector values corresponding to the heterozygous parent are opposite each other in the two states a and b. These two states therefore have equivalent inheritance vectors and, because state b has fewer recombinations, it is retained and state a is eliminated.

Ambiguous Inheritance Values and Fully Informative for One Parent Loci

Ambiguous inheritance values complicate the handling of fully informative for one parent loci. At this locus, we apply an optimization to propagate the inheritance vector bits for the homozygous parent from the previous locus. This requires only copying in the case of unambiguous inheritance values, and results in two equivalently labeled states.

The situation is different when a previous state has children with ambiguous inheritance values. In that case, the corresponding two inheritance vectors that Hapi builds are not equivalent because, for children with ambiguous inheritance values, the homozygous parent's inheritance bits are opposite one another rather than equivalent. At the same time, the homozygous parent's inheritance bits for any unambiguous children remain identical across the two values.

Consider the example in Figure 2-5, which is modified from the example in Figure 2-4 to include ambiguous inheritance values. As usual, the inheritance vector values for the heterozygous parent are opposite in the two states. However, the ambiguous inheritance bits correspond to two entirely opposite values, so the two resulting states do not have identical inheritance vector values for the homozygous parent (we underline these differing values in the figure). Because the two inheritance vectors are not equivalent, the algorithm cannot eliminate one of the two states. Even so, because the heterozygous parent's inheritance values are still exactly opposite, if the next locus is fully informative for the other parent, Hapi can produce one state at that locus. Note that although these two inheritance vectors are not equivalent, other previous states may map/transition to states that are equivalent, thereby enabling the elimination of some states. In other words, the number of states does not always double when there are ambiguous inheritance values.

2.5.3 Partly Informative Loci

Partly informative loci (and children with missing data; see Section 2.8) are the primary reason the haplotyping problem for families is challenging. This locus type came up earlier in the context of the ambiguous inheritance vector values optimization (see Section 2.4.2).

Processing partly informative loci consists of deducing the inheritance vector values for any homozygous children and applying the ambiguous inheritance values optimization for the heterozygous children. Hapi determines, for each of the four parents' phase assignments, which homolog the parents necessarily transmitted to their homozygous children. It also determines the two valid inheritance vector values for the heterozygous children and applies the aforementioned optimization by mapping the states at the previous locus to the current locus.

Whereas the number of states would otherwise always be $4 \cdot 2^h$, this optimization limits the number of states to be less than or equal to $4n$ where n is the number of states at the previous locus. While generally this number could grow unwieldy from successive partly informative loci, there actually exists a bound on the number of states that can result across a series of partly informative loci (see Section 4.3.2). Section 4.3 analyzes Hapi's runtime complexity in detail.

2.5.4 Uninformative Loci

Uninformative loci are homozygous for both parents and therefore give no information about recombinations. As such, Hapi does not build states at these loci, and when the system analyzes the succeeding locus, it uses the states from the most recent informative locus. The system does resolve the children's phase at these loci, deducing which is consistent with the parents' genotypes and assigning the children's alleles to homologs as needed.

2.6 Initial State

To build the initial state from which to haplotype a given chromosome, Hapi uses either a fully informative for both parent locus or two loci that are fully informative for opposite parents. Beginning at the first locus on the chromosome, Hapi looks for these types of loci to define the initial state. It skips any partly informative loci it encounters before the initial state is defined. Later, after defining an initial state and haplotyping the remainder of the chromosome, Hapi resolves haplotypes at these early partly informative loci by performing reverse haplotyping starting from the locus that established the initial state. Uninformative loci do not depend on nor produce states, so the system phases these loci whenever it encounters them.

It is straight forward to define an initial state using a locus that is fully informative for both parents. This locus type has exactly four possible inheritance vectors, and deducing these vectors does not require any previous state. The choice of which of the four inheritance vectors to use in the initial state is arbitrary, since, as discussed earlier, the inheritance vectors are equivalent. Different choices result in the same haplotypes assigned to different homologs in the parents. That is, the parents' haplotypes are merely labeled differently, but otherwise the results are identical.

A fully informative for one parent locus defines half of an inheritance vector, giving information only for the bits that correspond to the heterozygous parent. This locus type has two possible values for that portion of the inheritance vector, and just as before, Hapi chooses between the two arbitrarily. The choice affects only the homolog labels for the heterozygous parent's haplotypes. The initial state thus becomes partially defined with values for the heterozygous parent. Later, when the system encounters a locus that is fully informative for the undefined parent (or a locus fully informative for both parents), it fills in the inheritance vector values for the undefined parent, and haplotyping proceeds forward normally from this point. The system handles any intervening loci that are fully informative for the already-defined parent in the normal way, while still leaving the homozygous parents' inheritance vector bits undefined.

Figure 2-6 (described in more detail in Section 2.9) gives an example of an initial state defined from two fully informative loci (numbered 8 and 12).

2.7 Handling Ambiguities

As noted earlier, there is the possibility for ambiguities to arise—i.e., for more than one state at a given locus to produce a minimum-recombinant haplotype assignment. There are two ways this can occur. First, a fully informative locus that would otherwise produce a single output state (starting back tracing) may have two or more states that result in the same (minimal) number of recombinations. In this case, both states are equally valid, and back tracing must consider both. The second type of ambiguity occurs during back tracing, when a given minimal state has pointers to multiple previous states states that each yield minimal recombinations.

In both of these cases, there are paths through multiple states at some number of loci that are all minimally recombinant, and Hapi must consistently account for these during back tracing. Note that, with the exception of partly informative loci, the parents' genotypes uniquely infer the children's haplotypes. Ambiguity in states that produce minimal recombinations therefore apply only to the haplotypes of the parents and to heterozygous children at partly informative loci. To address ambiguities, the algorithm first arbitrarily chooses one of the minimal states at the ambiguous locus and assigns its haplotype values. The system then marks individuals' haplotypes ambiguous according to the differences in the states. If the phase for only one parent differs between the states, the algorithm marks that parent as ambiguous. If the phase for both parents differ between the states then Hapi marks both parents as ambiguous. At partly informative loci, if either parent is ambiguous, the heterozygous children's phase is also ambiguous and the system marks them as such.

After marking the necessary individuals as ambiguous, the system proceeds to the previous locus, tracking all minimum recombinant paths to the locus. It then follows the same procedure to assign any ambiguities with one change. Instead of arbitrarily choosing a state to apply at this previous locus, it applies a state that leads to the state assigned at locus the algorithm just visited. This ensures that the end result is a consistent haplotype assignment (i.e., that it follows a valid path through the states across loci).

2.8 Missing Data

Missing genotype data can result either because of quality control mechanisms associated with genotyping technologies or because of non-Mendelian errors (which can be removed using various software packages [28, 25]). It is straight forward to handle loci that have children with missing data. Hapi simply copies the inheritance vector values corresponding to the child with missing data from the state(s) at the previous locus to the newly built states at the current locus. This approach assumes a lack of recombination for that child, which suffices for the same reason that assuming no recombination at loci where a parent is homozygous does. Because the inheritance vector values for that child will no longer be opposite each other between states, but will instead be identical, Hapi cannot eliminate states in the way it does when no data is missing. However, it is possible to eliminate states in most cases. The following constitutes one of Hapi's optimizations, in addition to the five mentioned in Section 2.4.

Consider a set of states that have equivalent inheritance vectors when the missing data children are ignored and with identical inheritance values for those missing data children (i.e., states built based on the same previous state). Let x be the number of children with missing data, and let r be the value of the smallest number of recombinations among this set of states. The states in this set are x or $2x$ recombinations away from having equivalent inheritance vectors, depending on whether the inheritance values are opposite each other for transmissions from one or both of the parents. (Viewed another way, if two states have the same assignment of alleles to homologs for one parent and opposite assignments for the other, the inheritance vectors are x recombinations away from being equivalent. If both parents have opposite allele assignments, the inheritance vectors must be entirely opposite each other and therefore $2x$ recombinations separate them: the missing data children's inheritance values are identical, not opposite.) Considering states that are separated by x recombinations, a state that has more than $r + x$ recombinations will always be less optimal than the minimal state and can therefore be removed. Even if all the missing data children later recombine relative to the state with r recombinations, which would produce an inheritance vector equivalent to the larger state, that minimal state would yield $r + x$ recombinations—i.e., fewer than that for the larger state.

Although this technique will not always eliminate the same number of states as if full data were available, it is quite effective. Our experimental results demonstrate this as Hapi very efficiently analyzes a real dataset that includes missing data (see Results). Often one state at a locus will have zero or one recombinations compared to another state that has all or all but one child recombining. In such a case, the technique just described can eliminate the state with more recombinations.

Hapi does not currently handle loci that are missing data for one or both parents. Chapter 6 describes how it can be extended to do so.

2.9 Example

We give a brief example illustrating some aspects of our algorithm in Figure 2-6. This example is from real data for one of the families in the Huntington's Disease Venezuela Collaborative Study [9] dataset discussed in Results. The initial locus 8 defines inheritance vector values for parent 1, the heterozygous parent, but leaves the values for parent 0 undefined (designated by −). When analyzing this example, Hapi produces a complete initial state at locus 12, where it deduces inheritance vector values for parent 0 and copies those for parent 1 from locus 8. (Note: this figure omits uninformative loci.)

Locus 14 is partly informative, and with one state at the previous locus, it has only four states corresponding to the four possible parents' phase assignments. The figure shows two of these four states, one on the left and one on the right. The two omitted states have four and five recombinations at locus 14 and still more at locus 16 and 17.

The left side state at locus 14 has two recombinations. It transitions to two states at locus 16, one with a total of three recombinations and one with five; the figure shows the state with fewer recombinations. These two states at locus 16 each transition to the same two states at locus 17, and we include the one with fewer recombinations in the figure.

The right side state for locus 14 has three recombinations. Although this is greater than the two local recombinations shown for the left side state, this state actually yields fewer recombinations globally. It transitions to two states at locus 16, one of which produces no additional recombinations, and likewise that non-recombinant state produces zero recombinations at locus 17. This path of states therefore has only three recombinations, which is minimal across these loci.

Although the above analysis considered the downstream effects of each state at locus 14 separately, Hapi considers all states at successive loci at the same time and does not revisit each locus. The four states at locus 14 each transition to two non-equivalent (because of ambiguous inheritance values) states at locus 16, for a total of eight states. Because locus 16 is fully informative for parent 1, the inheritance vector values for that parent are equivalently labeled in these states. Locus 17 is heterozygous for parent 0 and produces exactly four equivalently labeled states, and the state with the fewest recombinations must be globally minimal. This globally minimal state is on the right side of the figure.

Locus		p_0	p_1	c_0	c_1	c_2	c_3	c_4	# Rec
8	*hap* $\vec{v}$	⟨a, a⟩	⟨a, c⟩	⟨a, a⟩ ⟨−, 0⟩	⟨a, c⟩ ⟨−, 1⟩	⟨a, c⟩ ⟨−, 1⟩	⟨a, c⟩ ⟨−, 1⟩	⟨a, a⟩ ⟨−, 0⟩	0
12	*hap* $\vec{v}$	⟨g, t⟩	⟨t, t⟩	⟨t, t⟩ ⟨1, 0⟩	⟨t, t⟩ ⟨1, 1⟩	⟨g, t⟩ ⟨0, 1⟩	⟨t, t⟩ ⟨1, 1⟩	⟨t, t⟩ ⟨1, 0⟩	0

14	*hap* $\vec{v}$	⟨c, a⟩	⟨c, a⟩	⟨a, c⟩ ⟨1, 0⟩	⟨a, a⟩ ⟨1, 1⟩	⟨c, c⟩ ⟨0, **0**⟩	⟨a, c⟩ ⟨1, **0**⟩?	⟨a, c⟩ ⟨1, 0⟩	2	⟨c, a⟩	⟨a, c⟩	⟨a, c⟩ ⟨1, **1**⟩?	⟨a, a⟩ ⟨1, **0**⟩	⟨c, c⟩ ⟨0, 1⟩	⟨a, c⟩ ⟨1, 1⟩	⟨c, a⟩ ⟨**0**, 0⟩?	3
16	*hap* $\vec{v}$	⟨a, a⟩	⟨g, a⟩	⟨a, g⟩ ⟨1, 0⟩	⟨a, a⟩ ⟨1, 1⟩	⟨a, g⟩ ⟨0, 0⟩	⟨a, g⟩ ⟨1, 0⟩	⟨a, a⟩ ⟨1, **1**⟩	3	⟨a, a⟩	⟨a, g⟩	⟨a, g⟩ ⟨1, 1⟩	⟨a, a⟩ ⟨1, 0⟩	⟨a, g⟩ ⟨0, 1⟩	⟨a, g⟩ ⟨1, 1⟩	⟨a, a⟩ ⟨0, 0⟩	3
17	*hap* $\vec{v}$	⟨t, c⟩	⟨c, c⟩	⟨c, c⟩ ⟨1, 0⟩	⟨c, c⟩ ⟨1, 1⟩	⟨t, c⟩ ⟨0, 0⟩	⟨c, c⟩ ⟨1, 0⟩	⟨t, c⟩ ⟨**0**, 1⟩	4	⟨t, c⟩	⟨c, c⟩	⟨c, c⟩ ⟨1, 1⟩	⟨c, c⟩ ⟨1, 0⟩	⟨t, c⟩ ⟨0, 1⟩	⟨c, c⟩ ⟨1, 1⟩	⟨t, c⟩ ⟨0, 0⟩	3

Figure 2-6: An example from one of the families in the Huntington's Disease Venezuela Collaborative Study [9] dataset discussed in Chapter 4. The loci are from chromosome 3 and we number them sequentially in the order they occur physically. For simplicity and conciseness, we omit uninformative loci and one non-recombinant fully informative locus for parent 0 between 8 and 12. Bold inheritance vector values designate recombinations. Each state/row lists its total number of recombinations. Note that the state at locus 14 with minimum recombinations is ultimately not minimum-recombinant globally. Section 2.9 contains a detailed discussion of the example in this figure.

Chapter 3

Maximum Likelihood Haplotyping

We now formulate the problem of maximum likelihood haplotyping and show how to solve it using the same techniques as those we employ for minimum-recombinant haplotyping. Our maximum likelihood approach expands on the minimum-recombinant techniques in order to efficiently process and represent the more complex haplotype possibilities for maximum likelihood haplotyping.

3.1 Maximum Likelihood Haplotyping Equation

Suppose we have genotyped loci numbered $0 \ldots L$ for each member of a nuclear family with c children, and assigned inheritance vectors $\vec{v}_l$ for each locus l. Let θ_l be the user-provided recombination frequency (i.e., probability of recombination) between locus l and $l-1$ for all $0 < l \leq L$. Also let $r(l) = H(\vec{v}_{l-1}, \vec{v}_l)$ or the number of recombinations (hamming distance) between the inheritance vectors at loci $l-1$ and l. Then the probability of the assigned inheritance vectors is:

$$\mathcal{P} = \prod_{l=1}^{L} \theta_l^{r(l)} \cdot (1-\theta_l)^{(2c-r(l))}. \tag{3.1}$$

Using log likelihoods, this can be written as:

$$\mathcal{L} = \sum_{l=1}^{L} \ln(\theta_l) \cdot r(l) + \ln(1-\theta_l) \cdot [2c - r(l)]. \tag{3.2}$$

This formulation of the maximum likelihood problem shows clearly the relationship of the maximum likelihood problem to the minimum-recombinant one. If all loci have the same recombination frequency $\theta < 0.5$, then the maximum likelihood solution is the same as the minimum-recombinant one since $\ln(\theta) < \ln(1-\theta)$ across all loci, so decreased $r(l)$ values increase the overall likelihood. However, when the recombination frequencies differ across loci, more recombinations at one locus may have higher likelihood than fewer recombinations at another.

A dynamic programming equation computing maximum likelihood haplotypes can be written as follows, where l is a locus, and $\vec{v}, \vec{w}$ are inheritance vectors:

$$P(l, \vec{v}) = \max_{\vec{w}} \{P(l-1, \vec{w}) \cdot \theta_l^{H(\vec{w},\vec{v})} \cdot (1-\theta_l)^{(2c-H(\vec{w},\vec{v}))}\}. \tag{3.3}$$

Using log likelihoods, the dynamic programming formulation becomes:

$$L(l, \vec{v}) = \max_{\vec{w}} \{L(l-1, \vec{w}) + \ln(\theta_l) \cdot H(\vec{w}, \vec{v}) + \ln(1-\theta_l) \cdot [2c - H(\vec{w}, \vec{v})]\}. \tag{3.4}$$

Immediate application of the above formula is problematic because we cannot completely ignore uninformative loci: they have non-zero recombination frequencies that affect the overall probability of a solution. Without some novel insight, it is necessary to model most or all of the $2^{2c}/4$ non-equivalent inheritance vectors at uninformative loci, and at least $2^c/2$ inheritance vectors at loci that are fully informative for one parent.

3.2 Recombinations at Uninformative Loci

In order to account for recombination frequencies at loci where both parents are homozygous—i.e., at uninformative loci—Hapi computes modified recombination frequencies at all other informative loci, including fully informative for one parent loci where one parent is homozygous. (Section 3.3 discusses how we handle potential recombinations at homozygous parent at fully informative for one parent loci.) These modified recombination frequencies include the recombination frequencies for all uninformative loci that occur between a given informative locus and the nearest upstream informative locus. In calculating these probabilities, the algorithm is pre-computing the effects of recombination frequencies at uninformative loci, allowing it to avoid directly processing such loci.

We denote Hapi's modified recombination frequency at a locus l as ϕ_l and the frequency of non-recombination (expressed above as $(1-\theta_l)$) as ψ_l. To calculate ϕ_l and ψ_l for a locus l, let l_0 be the nearest upstream informative locus and let $l_1, \ldots, l_{n-1}$ be the loci uninformative for parent p that appear between l_0 and l. Let $l_n = l$ and let l^* be the locus with the highest recombination frequency, i.e., find $l^* \in \{l_1, \ldots, l_n\}$ such that $\theta_{l^*} = max_{i=1}^{n} \theta_{l_i}$. Then:

$$\phi_l = \theta_{l^*} \cdot \prod_{i=1, l_i \neq l^*}^{n} (1 - \theta_{l_i}). \tag{3.5}$$

Thus, the probability of recombination between locus l_0 and $l_n = l$ is equal to the maximum between-marker recombination frequency within the region spanned by these loci, or θ_{l^*}, multiplied by the probability of not recombining anywhere else. Note that θ_{l^*} is the probability of recombining between locus l^* and $l^* - 1$, and either or both of these loci can be uninformative. Hapi stores the locus number l^* so that the final haplotype solution includes any recombinations in their most likely positions.

A consequence of this formula is that at most one recombination can occur between any two informative loci on a given homolog. Thus, within a region of uninformative loci, we do not model the possibility of intervening gene conversions or double recombinations. Not modeling such events make sense because it is impossible to observe or verify them. Furthermore, haplotypes that include additional recombinations or gene conversions not directly implied by the data are less likely than those without these events since $\theta_l < 0.5$ means recombination is less likely than non-recombination. Therefore, even if we were to model such events, they would not ultimately appear in the haplotype solution, so we lose nothing by not modeling them.

The probability of not recombining between locus l_0 and $l_n = l$ is the product of non-recombination across each of the locus intervals:

$$\psi_l = \prod_{i=1}^{n} (1 - \theta_{l_i}). \tag{3.6}$$

The equations for ϕ_l and ψ_l utilize the recombination frequencies between each pair of loci rather than a single recombination frequency spanning the region between l_0 and l. This is the case because the haplotyping output must place every recombination at some discrete location between a pair of markers. There must exist a pair of markers flanking every recombination, and sometimes one or both of these will be uninformative. This matches the maximum likelihood approach employed by other algorithms, which calculate the probability of recombining (or not) between each pair of markers, not just those that are informative. A consequence of this formulation is that $\phi_l + \psi_l \neq 1$. This occurs because, as we earlier noted, these probabilities account for the possibility of only one recombination on a given homolog between any two informative loci—more than one recombination will always be less likely. Some applications—notably linkage analysis (see Section 5.1)—may benefit from using a single recombination frequency between the region spanned by l_0 to l. Our algorithm functions the same regardless of how we calculate ϕ and ψ.

To increase numerical stability and efficiency, Hapi uses the log likelihood formulation of this dynamic programming problem. This formula substitutes multiplication for exponentiation and uses the values $\ln(\phi_l)$ and $\ln(\psi_l)$ which requires summation instead of multiplication to calculate. The dynamic programming equation for maximum log likelihood haplotypes at a locus l informative for parent p is then given by the following:

$$L(l, \vec{v}) = \max_{\vec{w}} \{ L(l-1, \vec{w}) + \ln(\phi_l) \cdot H(\vec{w}, \vec{v}) + \ln(\psi_l) \cdot [c - H(\vec{w}, \vec{v})] \} \tag{3.7}$$

3.3 Complications Due to Fully Informative for One Parent Loci

Hapi's minimum-recombinant haplotyping algorithm does not model recombinations at fully informative for one parent loci. For maximum likelihood haplotyping, it suffices to initially only model states at fully informative for one parent loci that do not exhibit recombination from the homozygous parent. If the child does not exhibit

(a)

Locus Type	Locus #	$\vec{v}$
FI	$l-1$	$\langle 0,0\rangle$
FI-P0	l	$\langle 0,0\rangle$
PI	$l+1$	$\langle 0,\mathbf{1}\rangle$?
FI	$l+2$	–

(b)

Locus Type	Locus #	$\vec{v}$
FI	$l-1$	$\langle 0,0\rangle$
FI-P0	l	$\langle \mathbf{1},0\rangle$
PI	$l+1$	$\langle 1,0\rangle$
FI	$l+2$	–

(c)

Locus Type	Locus #	$\vec{v}$
FI	$l-1$	$\langle 0,0\rangle$
FI-P0	l	$\langle \mathbf{1},0\rangle$
PI	$l+1$	$\langle 1,0\rangle$
FI	$l+2$	$\langle \mathbf{0},\mathbf{1}\rangle$

Figure 3-1: Possible assignments across a fully informative for one parent locus and partly informative locus yielding different likelihoods depending on where the recombinations are located and on downstream assignments.

recombination from that parent at the next locus where that parent is heterozygous, the lack of recombination at the earlier locus yields highest likelihood. However, when the later locus does exhibit recombination, placing the recombination before the earlier fully informative for one parent locus may increase likelihood.

A further complication to this issue of fully informative for one parent loci is the interactions between such loci and partly informative loci. Figure 3-1 illustrates with three possible inheritance value assignments across a series of loci. Here, the inheritance assignment at locus $l-1$ is constant and assumed fixed. Locus l is fully informative for one parent and $l+1$ is partly informative. If a child that is heterozygous at locus $l+1$ exhibits recombination relative to the inheritance vector at locus l, the system would normally mark the child's inheritance as ambiguous, as Figure 3-1(a) shows. Another possibility is to place a recombination from the homozygous parent at locus l, which Figure 3-1(b) shows. In this case, the recombination at locus $l+1$ no longer occurs. A further confounding factor is the inheritance assignments at downstream loci; as Figure 3-1(c) shows, later loci may exhibit recombination relative to the choice of lack of recombination at the partly informative locus.

Complex dependence between possible inheritance values also occur when a fully informative for one parent locus appears immediately after a partly informative locus with a heterozygous child.

3.4 Complications From Partly Informative Loci

Besides the issues that arise from fully informative for one parent loci, an unambiguous non-recombinant inheritance vector assignment for a heterozygous child at a partly informative locus may not always yield maximum likelihood. Figure 3-2 shows an example in which the default non-recombinant inheritance value is assigned at a locus l and some further downstream locus $l+x$ recombines on both homologs. Because recombination frequencies differ across loci, if the child is heterozygous at all intervening partly informative loci, it may be more likely place the recombinations on both homologs upstream at locus l.

Locus Type	Locus #	$\vec{v}$
FI	$l-1$	$\langle 0,0\rangle$
PI	l	$\langle 0,0\rangle$
PI	$\vdots$	$\vdots$
FI	$l+x$	$\langle \mathbf{1},\mathbf{1}\rangle$

Figure 3-2: Example in which moving the recombinations on both homologs upstream to an earlier partly informative locus may increase the overall likelihood.

3.5 Tracking Alternate Inheritance Assignment Likelihoods

In order to correctly model the maximum likelihood inheritance assignments, Hapi tracks the likelihood of switching to the *alternate* inheritance assignment for each child at each state. The alternate inheritance assignment for a child at a fully informative for one parent locus switches the inheritance value for the homozygous parent. The maximum likelihood alternate inheritance at a given locus may include alternate inheritance assignments at further upstream loci, and the system must track the likelihoods for the most likely assignment. Let $\rho_l = \phi_l/\psi_l$, the relative likelihood of recombining at locus l. This value divides out ψ_l since the likelihood of the state includes this value of not recombining; later modifications to the likelihood of a path must divide out this value and substitute the probability of recombination.

The following equation governs the probability of switching to an alternate inheritance at a locus that is fully fully informative for parent p and homozygous for the other parent q:

$$Alt_l = \begin{cases} \rho_l & \text{if locus } l-1 \text{ is fully informative for parent } q \\ \max\{\rho_l; Alt_{l-1}\} & \text{if locus } l-1 \text{ is fully informative for parent } p \\ \rho_l & \text{if locus } l-1 \text{ is partly informative and does} \\ & \text{not recombine to locus } l \\ & \\ \max\{\rho_l; Alt_{l-1}/\rho_l\} & \text{if locus } l-1 \text{ is partly informative and} \\ & \text{recombines relative to locus } l \\ & \\ Alt_{l-1} & \text{if child is missing data at } l \end{cases} \tag{3.8}$$

Note that loci that we consider loci that are partly informative but where the child is homozygous as fully informative for both parents.

The probability of switching to alternate inheritance for a child that is heterozygous or missing data at a partly informative locus is given as follows:

$$Alt_l = \begin{cases} 1 & \text{if the previous locus is not fully informative for one parent and the child recombines: ambiguous inheritance} \\ 2 \cdot \rho_l & \text{if the previous locus is fully informative for both parents and does not recombine} \\ \max\{2 \cdot \rho_l ; Alt_{l-1}\} & \text{if the previous locus is partly informative with the child heterozygous and does not recombine} \\ \max\{2 \cdot \rho_l ; \rho_l / Alt_{l-1}\} & \text{if the previous locus is fully informative for one parent, exhibits recombination, and } (Alt_{l-1}/\rho_l) > 1 \\ 2 \cdot \rho_l & \text{if the prevoius locus is fully informatie for one parent, exhibits recombination, and } (Alt_{l-1}/\rho_l < 1) \\ 1 & \text{if the previous locus is fully informatie for one parent, does not recombine, and } (Alt_{l-1} \cdot \rho_l > 1)\text{: introduces ambiguous inheritance} \\ \max\{2 \cdot \rho_l ; Alt_{l-1} \cdot \rho_l\} & \text{if the prevoius locus is fully informatie for one parent, does not recombine, and } (Alt_{l-1} \cdot \rho_l < 1) \end{cases} \tag{3.9}$$

These equations give likelihoods for a single state's alternate inheritance probabilities relative to the previous state. Because multiple states may map to the same state at the next locus, and because the alternate inheritance values may vary across these states, the system must track both a minimum and a maximum alternate probability at each state, corresponding to the minimum and maximum alternate probabilities for the previous states.

When the system determines that it is more likely to use the alternate inheritance at some previous locus in order to reach a given state, it incorporates the alternate inheritance probability into the state's overall probability. Since there is both a minimum and maximum alternate inheritance probability that differ in general, the overall state probability is ambiguous and is given as a minimum and maximum value. Tracking the probability range allows Hapi to remove edges to previous states who maximum probability is less than the minumum probability of some other previous state. Figure 3-3 shows example states with minimum and maximum probabilities

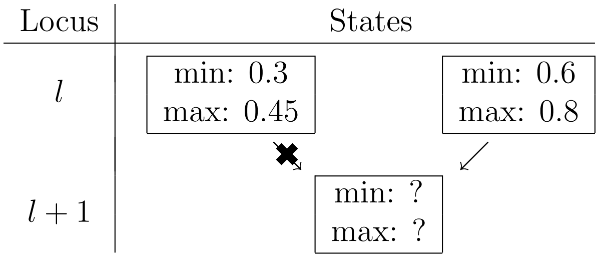


Figure 3-3: Two example states with minimum and maximum probabilities at a locus l that transition to a particular state at locus $l+1$. Because the maximum probability for the state on the left is less than the minimum probability for the state on the right, the system does not keep a pointer to that state from the state at $l+1$.

at a locus l. The left side state has a lower maximum probability than the minimum probability of the right side state. In a case such as this, Hapi will not keep a pointer to the less likely previous state in the state at the next locus that each maps to. Hapi's minimum and maximum probabilities give an approximation of a state's or path's actual probability. Eventually the system performs back tracing to resolve the ambiguous children's probability values.

Sometimes the maximum alternate inheritance probability yields greater likelihood while the minimum alternate inheritance probability does not. In this case, the system stores the fact that alternate inheritance upstream is ambiguous and back tracing later determines the overall probabilities per path.

Back tracing yields fixed probabilities for states at some upstream locus. To assign the maximum likelihood path of states, Hapi locates the state at the upstream locus that has maximum probability. It then performs forward tracing to assign the path of states that yields this probability. Forward tracing requires that Hapi store pointers to the state at the next locus that has the highest likelihood. These pointers are analogous to the pointers to states at the previous locus. While performing back tracing, Hapi stores the state (or states) at the next locus the yields the highest probability at the current state. Because of alternate inheritance, this will often include several states at the next locus with differing alternate probabilities to be applied at the previous locus. The algorithm can only compare next states whose alternate inheritance to be used at upstream loci are identical.

3.6 Forward Tracing

Back tracing finishes at the most upstream locus that has only one state, which will have had its haplotype values assigned. At this point, Hapi will have computed full probabilities—including any alternate inheritance assignments—for states at downstream loci. The probabilities stored in states at the next locus include the probabilities of states at all downstream loci. Hapi locates the state at the next locus with maximum likelihood and performs forward tracing from this point. As noted, states contain pointers to the next downstream locus that has maximum likelihood to enable this forward tracing process. Hapi applies the haplotype values stored in each state—including applying any alternate inheritance assignemnts—as it processes each.

Chapter 4

Results

Hapi is a novel algorithm and program for haplotyping nuclear families that performs orders of magnitude faster than related programs. In this chapter we present experimental results comparing Hapi to existing state-of-the-art algorithms. We also describe Hapi's output formats and give a formal complexity analysis of the algorithm.

4.1 Runtime Comparison to Other Family-Based Haplotyping Algorithms

We have evaluated Hapi's runtime performance compared to three state-of-the-art algorithms: Merlin [1], Allegro [11], and Superlink [6], programs in current use for family-based haplotype assignment. Like most algorithms for computing maximum likelihood haplotypes, these programs have exponential complexity in general. However, each contains several optimizations, and these are the most suitable programs for comparison to Hapi. We omitted GENEHUNTER from our comparison because Merlin outperforms it [1].

We ran each program on a dataset of nuclear families derived from a pedigree from the Huntington's Disease Venezuela Collaborative Study [9]. This Venezuelan pedigree has 757 individuals and 458 families. None of Merlin, Allegro, or Superlink can successfully haplotype such a large pedigree. Hapi can currently only analyze nuclear families where both parents have genotype data, so the pedigree was broken up into such families. The choice to break up such a large pedigree into smaller sets of related individuals is necessary regardless of which haplotyping tool is used since runtime and memory requirements impose hard limits on the scalability of existing algorithms.

The derived nuclear family dataset contains 103 nuclear families where both parents have data. These families have a total of 438 individuals. Note that because we analyzed the families separately, we double counted individuals that appear in more than one family (e.g., as a parent in one and a child in another, or as a parent in more than one family).

These families range in size from 1 to 11 children, with an average of 2.23 children per family. There are 86 families with 3 or fewer children (308 total individuals), with

2.30 GHz AMD Opteron				
Program	All Families		≤ 3 Children	
	Runtime	Speedup	Runtime	Speedup
Hapi	3.112s	–	2.255s	–
Merlin	1005s	323x	8.662s	3.84x
Allegro v2	7661s	2462x	14.50s	6.43x
Superlink	1393s*	448x	38.75s	17.2x

Figure 4-1: Runtimes for maximum likelihood haplotyping using Hapi, Merlin, Allegro and Superlink of nuclear families from the Huntington's Disease Venezuela Collaborative Study [9]. We list times for haplotyping all nuclear families and for haplotyping those with three or fewer children. *Superlink failed to haplotype the family with 11 children; we therefore used only 8 of the children from the 11 child family to time it. Times are averages from running Hapi eight times and Merlin, Allegro, and Superlink three times each.

an average of 1.56 children for that subset of families. Using the Illumina linkage IV_v3 SNP panel, genotypes at 5,456 SNPs covering the whole genome were obtained for each individual in the dataset [9]. The numbers of SNPs per chromosome are roughly proportional to the chromosome's size and range from 102 on chromosome 21 to 468 on chromosome 2. Prior to analysis, the PEDSTATS [28] and PedCheck [25] programs were used to remove genotypes exhibiting non-Mendelian errors.

Figure 4-1 shows timing results from our experiments. The times are for performing maximum likelihood haplotyping using Hapi, Merlin, Allegro v2, and Superlink on a 2.30GHz AMD Opteron machine with 32GB of RAM. Although this is a multi-core processor, none of the algorithms are parallelized, so their runtimes are directly comparable. We used Hapi to infer maximum likelihood rather than minimum-recombinant haplotypes in this set of experiments because the other programs address that problem, and because that form of haplotyping is slower in Hapi. All programs except for Superlink (see below) used less than 8GB of memory.

Superlink ran for over 6 hours without finishing when we used it haplotype chromosome 1 for all families in the dataset. At that time, the program reported that 0% of the haplotyping was complete. We found that Superlink uses an excessive amount of memory (¿24GB) to haplotype a family with 9 or 10 children. The times for Superlink therefore reflect its haplotyping a modified set of families, with three of the children removed from the original 11 child family. Superlink used less than 8GB of memory when analyzing this modified dataset.

We include times for haplotyping all families in the dataset (modified for Superlink), as well as the subset of families with three or fewer children in Figure 4-1. Because of the fixed and disproportionate overhead involved in printing the haplotypes in Hapi and Merlin (∼.5 seconds in Hapi or about 16% of runtime and ∼29 seconds in Merlin or <3% of runtime), we report the times only for reading in the dataset and performing the haplotyping in these programs, but not printing the results. Source code is not publicly available for Superlink, so we could not modify it to avoid printing haplotypes, but such a change is unlikely to dramatically affect its

1.40 GHz Pentium M				
Program	All Families		$\leq$ 3 Children	
	Runtime	Speedup	Runtime	Speedup
Hapi	4.732s	–	3.451s	–
PedPhase 2.0	> 21600s (6h)†	> 4500x	> 21600s (6h)†	> 6000x

Figure 4-2: Runtimes for minimum-recombinant haplotyping of the nuclear family dataset derived from the Huntington's Disease Venezuela Collaborative Study [9]. PedPhase 2.0 runs in Windows and we consequently used a different machine to obtain these times than the one used for the times listed in Figure 4-1. † PedPhase failed to haplotype chromosome 1 in over 6 hours. Times for Hapi are averages from eight runs.

runtime. We also did not modify Allegro to prevent it from printing haplotypes, but its runtime also would not change significantly compared to the current results.

As Figure 4-1 shows, Hapi is substantially faster than Merlin, running 323 times faster for the entire dataset and 3.84 times faster for the subset of families with three or fewer children. Hapi compares even more favorably against Allegro and Superlink, even while Superlink can only haplotype a reduced-sized dataset. When haplotyping the entire dataset, Hapi runs 2462 times faster than Allegro and 448 times faster than Superlink's analysis of the modified dataset. For haplotyping the subset of families with three or fewer children, Hapi runs 6.43 times faster than Allegro and 17.2 times faster than Superlink.

Hapi's speedup for the entire dataset demonstrates experimentally the vast difference between the theoretical complexity of these algorithms. Whereas Merlin, Allegro, and Superlink have exponential runtime complexity, Hapi runs in polynomial time in practice (see Section 4.3). At the same time, the more modest gains for the families with three or fewer children is unsurprising. The other algorithms scale exponentially in the number of non-founders or, in the case of nuclear families, in the number of children in the family being analyzed. When that number is very small, an exponential algorithm will not differ as significantly from one that has polynomial runtime in practice. Our algorithm is still significantly faster than these programs even in this case that is less taxing to an exponential algorithm.

We evaluated the number of states Hapi produces for the entire dataset and listed the average number of states for the locus types in Figure 2-3. Fully informative for one parent loci produce and average of only 1.87 states, while partly informative loci have an average of only 6.31 states. This low number of states is further evidence of Hapi's efficiency.

4.1.1 Minimum-Recombinant Haplotyping Comparison

Besides these maximum likelihood systems, we compared Hapi's minimum-recombinant haplotyping to PedPhase 2.0, which uses an Integer Linear Programming algorithm to calculate minimum-recombinant haplotypes for pedigrees [18]. PedPhase 2.0 runs only in Windows, and we used a 1.40GHz Pentium M laptop with 1.24 GB of RAM

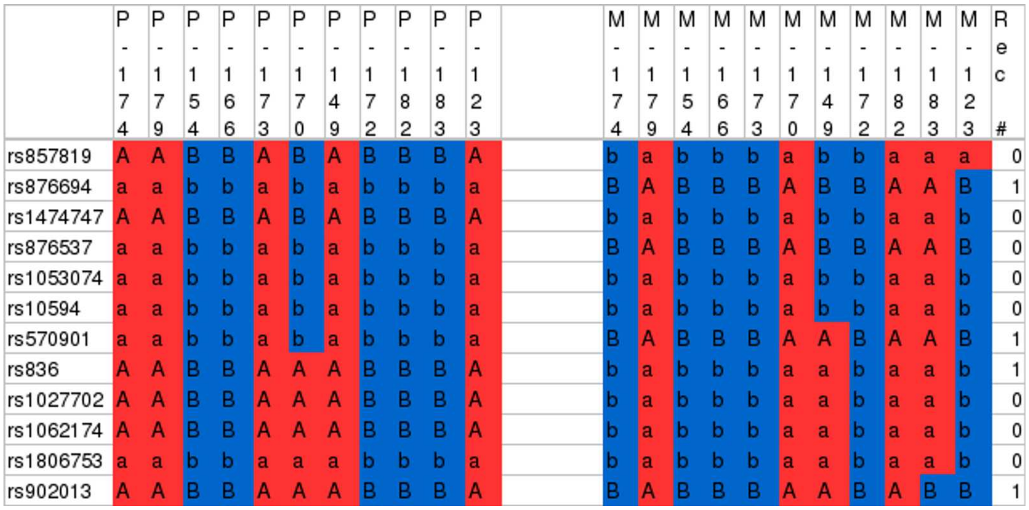
Total % Missing	Simulation Probability	Runtime	Slowdown	Speedup vs. Merlin
5%	3.83%	3.274s	5.21%	306x
10%	8.83%	3.564s	14.5%	281x
20%	18.8%	4.567s	46.8%	220x
30%	28.8%	6.897s	122%	145x
40%	38.8%	11.36s	265%	88.5x
50%	48.8%	36.38s	1070%	27.6x

Figure 4-3: Hapi's runtime performance for haplotyping the dataset discussed in Results in the presence of various total propositions of missing data. Because this dataset contains 1.17% missing data already, we dropped the marker data according to the indicated probabilities in order to obtain the total overall proportions of missing data. The figure lists the runtime, percentage slowdown compared to running Hapi on the unmodified dataset, and the speedup of Hapi compared to Merlin on these simulated datasets.

to compare the runtimes of these two systems. Figure 4-2 gives timing results on this machine for Hapi and PedPhase. We ran PedPhase on the entire dataset and on the families with three or fewer children. In both cases, PedPhase did not exceed available memory, and ran for over 6 hours without haplotyping even chromosome 1. Because 464 of the 5,456 total SNPs reside on chromosome 1, we estimate that the total runtime for PedPhase on this dataset would be at least 70 hours. In contrast, Hapi completes haplotyping the entire dataset in 4.732 seconds (in Linux) on this machine.

4.1.2 Missing Data Simulations

As we discuss in the appendix, the number of states in Hapi is affected by the number and pattern of markers that are missing data. Our nuclear family dataset contains only 1.17% missing data. To explore the runtime performance of Hapi in the presence of moderate to significant proportions of missing data, we modified it to randomly drop various proportions of data. Figure 4-3 gives the results of our simulations. In the most extreme case of 50% missing data, Hapi's average runtime was 36.38s, which is still 27.6 times faster than Merlin. Real datasets will generally contain 5% or less missing data, and we probabilistically dropped 3.83% markers from the original data to obtain approximately 5% missing data. In this scenario, Hapi performed only 5.21% slower than for haplotyping the dataset without the added missing data. These results demonstrate that Hapi is robust to haplotyping data with significant proportions of missing data and performs very well for the more modest missing data proportions for which it is likely to be used.



	P-174	P-179	P-154	P-166	P-173	P-170	P-149	P-172	P-182	P-183	P-123		M-174	M-179	M-154	M-166	M-173	M-170	M-149	M-172	M-182	M-183	M-123	Rec#
rs857819	A	A	B	B	A	B	A	B	B	B	A		b	a	b	b	b	a	b	b	a	a	a	0
rs876694	a	a	b	b	a	b	a	b	b	b	a		B	A	B	B	B	A	B	B	A	A	B	1
rs1474747	A	A	B	B	A	B	A	B	B	B	A		b	a	b	b	b	a	b	b	a	a	b	0
rs876537	a	a	b	b	a	b	a	b	b	b	a		B	A	B	B	B	A	B	B	A	A	B	0
rs1053074	a	a	b	b	a	b	a	b	b	b	a		b	a	b	b	b	a	b	b	a	a	b	0
rs10594	a	a	b	b	a	b	a	b	b	b	a		b	a	b	b	b	a	b	b	a	a	b	0
rs570901	a	a	b	b	a	b	a	b	b	b	a		B	A	B	B	B	A	A	B	A	A	B	1
rs836	A	A	B	B	A	A	A	B	B	B	A		b	a	b	b	b	a	a	b	a	a	b	1
rs1027702	A	A	B	B	A	A	A	B	B	B	A		b	a	b	b	b	a	a	b	a	a	b	0
rs1062174	A	A	B	B	A	A	A	B	B	B	A		b	a	b	b	b	a	a	b	a	a	b	0
rs1806753	a	a	b	b	a	a	a	b	b	b	a		b	a	b	b	b	a	a	b	a	a	b	0
rs902013	A	A	B	B	A	A	A	B	B	B	A		B	A	B	B	B	A	A	B	A	B	B	1

Figure 4-4: Output from Hapi showing the inherited homologs on chromosome 1 for a family with 11 children from the Huntington's Disease Venezuela Collaborative Study [9]. Hapi produces CSV format output, which we imported into a spreadsheet. To color the cells, we used conditional formatting based on the homolog value transmitted. The output of inheritance vector values uses letters `A` and `B`. Lower-case letters indicate the transmitting parent is homozygous and the presence of recombination unknown. Each column is labeled with the child's numerical id with either a 'P' or an 'M' preceding it to indicate either paternal or maternal-derived homologs. The left most column gives the SNP rs numbers, and the right most column lists the number of recombinations across all children at the given locus.

4.2 Hapi's Output

Hapi produces output in text or CSV format, suitable for import into a spreadsheet. It can output either the actual haplotypes with allele values or the children's inheritance vector values. The latter is useful for inspecting the results of meioses, including recombination patterns. Figure 4-4 shows the inheritance vector output from Hapi for a family with 11 children, imported into a spreadsheet. This output uses letter symbols rather than bit values, with lower case letters indicating that the corresponding meiosis is uninformative. To help identify recombinations sites, we use the spreadsheet program's conditional formatting feature to color the cells based on which homolog the child received. The output from Merlin, Allegro, and Superlink provide the same information as Hapi, but each of these programs uses its own text-based format. We expect that many geneticists will find the ability to import Hapi's output into a spreadsheet to be more intuitive and more convenient than the output from other programs.

4.3 Complexity Analysis of Hapi

The experimental results in Section 4.1 demonstrate that Hapi is extremely efficient in practice: it runs orders of magnitude faster than existing algorithms. If L is the number of marker loci, s is the maximum number of states produced at any locus, and c is the number of children, Hapi's complexity is $O(L \cdot s \cdot c)$. This is so because the algorithm visits each of the L loci at most two or, for maximum likelihood haplotyping, three times (additional visits occur during back tracing/forward tracing). At each locus, Hapi builds $O(s)$ states, and each state has size $O(c)$. While Hapi is efficient in practice, there are two rare corner cases where $s = O(2^c)$. That is, the number of states can be exponential in the number of children in the family being analyzed.

Because of the nature of the requirements for these corner cases to occur, they are extremely unlikely to happen in practice. As well, if a series of loci do exhibit one of these cases, the number of states becomes exponential only transiently, because a later locus that is fully informative for both parents and with data for all the children (or two successive loci fully informative for opposite parents) will produce a single state. One necessary condition for this state blowup is therefore that the series of loci not contain such a locus (or loci) that produces a single state.

4.3.1 Missing Data

The first exponential case arises because of missing data. Consider a fully informative for one parent locus where the previous locus has n states. If one or more children have missing data at this locus, the number of states for this locus, which would otherwise be n (absent previously ambiguous inheritance values), can instead become $2n$. In order for this increase in states to occur, a large proportion of the children must have missing data, or a large proportion must recombine. Without these conditions, the optimization described in Section 2.8 would apply. These conditions are unlikely in practice since recombination is rare, but one can construct such a pathological input. Note that if a child that was missing data at some previous locus has data at the current locus, states that were added at the previous locus effectively merge.

The above properties indicate the possibility, however remote, for Hapi to produce an exponential number of states from loci with missing data. Consider a series of loci where each successive locus has missing data for the same set of children at the previous locus as well as missing data for one more child. (That is, the first locus has data for all children, the second missing data for child c_0, the third missing data for child c_0 and c_1, etc.) This scenario could lead to an exponential number of states, but only if the previously mentioned optimization does not apply. A more complicated scenario in which at least half of the children have missing data at the starting locus would defeat the optimization since the value of x mentioned in Section 2.8 would be large. However, such a scenario is extremely unlikely to occur in practice.

4.3.2 Ambiguous Inheritance Values

The second way an exponential number of states may occur comes through ambiguous inheritance values, which Hapi introduces at partly informative loci. Ignoring additional states that may arise because of missing data, the first encountered partly informative locus will have 4 states. Without any recombination, the maximum number of states that may arise across any number of partly informative loci is 12. This maximum of 12 states occurs because lack of recombination constrains children's inheritance values to be fixed relative to each other. With these constraints, for a particular assignment of parent's alleles to homologs, there are three possible inheritance values for heterozygous children (two non-ambiguous and one ambiguous). Since there are four phase assignments of the parents at these loci, there can be at most $4 \cdot 3 = 12$ states.

When recombination occurs in a child or children between two partly informative loci, the number of states at the second locus increases *additively*, with two additional states for each assignment of parent's alleles, (for a total of eight added states). The recombination changes the constraints on the possible children's inheritance values and effectively breaks the children into two classes. The children in one class exhibits recombination relative to the other class, but until the remainder of the algorithm completes, which of these two classes has fewer recombinations is unknown, so the system must track additional states (i.e., both classes). In general, since all c children can recombine, this can lead to at most $4(2c + 1)$ states, a polynomial number, so the number of states cannot become exponential across a series of partly informative type loci only.

The number of states can grow to be exponential through interactions between partly informative and fully informative for one parent loci. Whenever a locus has states with ambiguous inheritance values and occurs immediately before a fully informative for one parent locus, the number of states may double, as described in Section 2.5.2. The scenario in which an exponential number of states can occur is when there are a series of alternating partly informative and fully informative for one parent loci. At each partly informative locus, one of the children must recombine and become heterozygous, yielding an ambiguous inheritance value, and that child must remain heterozygous at all successive partly informative loci. The recombination results in an addition $4 \cdot 2$ states at each partly informative locus, followed by a doubling in the number of states at the fully informative for one parent locus. The number of states doubles c times for an exponential blowup.

The above scenario is extremely unlikely. It requires recombinations to occur in all the children, all at partly informative loci, and without encountering a locus or loci that produce a single output state. Moreover, after recombining, the children must be heterozygous across all partly informative loci. This is unlikely because homozygous and heterozygous genotypes are equally probable for children at such loci. Although this scenario could occur for a small number of children (say two or three), the likelihood decreases as the number of children increases. Thus, when an exponential blowup would be most problematic (for large c), it becomes even less likely.

To estimate the probability of this scenario occurring, let $\theta < 0.5$ be the recombination probability between each locus within some series of loci. Let L be the number of loci in this series, and assume that each partly informative locus is followed by a fully informative for one parent locus and vice versa. Then the probability of all c children recombining anywhere within this series of loci is $\theta^c \cdot (1-\theta)^{(L-1)\cdot c}$, since there are $L \cdot c$ transition probabilities across all loci and children, and c of these are recombinations. The likelihood of a child being heterozygous across a series of p partly informative loci is $\left(\frac{1}{2}\right)^p$. A set of children that are either all heterozygous or all homozygous at a partly informative locus and that do not recombine subsequently will all share genotypes at downstream partly informative loci, being all either heterozygous or homozygous. It suffices therefore to require only one recombined child to have the necessary heterozygous genotype; the others will be the same. The following is an upper bound on the probability for this scenario, since it doesn't account for the likelihoods of the locus types appearing in the required order:

$$P < \theta^c \cdot (1-\theta)^{(L-1)\cdot c} \cdot \sum_{i=c}^{L} \left(\frac{1}{2}\right)^i . \tag{4.1}$$

The summation allows for the children to recombine anywhere within the series of loci. This is an extremely low probability, even if θ is near 0.5. This is also a conservative estimate, because the children must have the proper inheritance vector values at the locus preceding the locus at which it recombines in order for a recombination to actually take place. One way to ensure that this is so is for the children to be homozygous at all partly informative loci upstream of the first recombination point. Another possibility is for the children to exhibit additional recombinations so that they have the proper inheritance value before the recombination point. Both of these scenarios have low probability that we do not accounted for in the probabilistic bound given above.

Chapter 5

Application to Linkage Analysis

As we noted in the introduction, haplotypes have many important applications. Haplotypes inferred for family-based datasets are often used to perform linkage analysis for disease gene mapping. Here, we describe how to extend Hapi to perform linkage analysis. We also outline how to impute genotype or sequence data for individuals in a family based on data acquired for one or more of the family members.

5.1 Linkage Analysis

The goal of linkage analysis is to find a genetic locus that co-segregates with a disease trait. Given a particular inheritance vector (i.e., haplotype assignment) for each locus, it is relatively straight forward to calculate the per-locus likelihood of linkage, either using a parametric or non-parametric scoring function [14]. The maximum likelihood inheritance vectors Hapi produces can be used for this purpose. However, linkage analysis is usually performed not for a particular inheritance vector at each locus, but using the probability distribution of all possible inheritance vectors at each locus.

Although Hapi does not currently perform linkage analysis, it can be adapted to do so. To accomplish this, the system need not perform back tracing, and cannot reclaim memory associated with any less-likely states, but must retain all states for the linkage analysis calculations. The algorithm must also explicitly calculate the probability of each inheritance vector/state it considers—the maximum likelihood haplotyping algorithm finds the maximum likelihood state for each locus, but does not calculate the exact probability of each state. More specifically, Hapi does not include in each state the probability of the downstream state path, but includes the probability of the upstream state path and the transition probability from the optimal state at the previous locus. To properly calculate a state's probability, system must also include the probability of the optimal next state, as well as the transition probability to that state. Once the algorithm finishes building states for all loci on a chromosome, it can follow pointers to previous states and incorporate the transition probability to the next state as well as that state's probability. Note that the first and last loci have no upstream or downstream loci to consider and the states at these loci therefore do not include the corresponding probability components.

With the probability distribution of locus states calculated in this way, one can calculate LOD scores or non-parametric Z scores by weighting the score of each inheritance vector by its probability at the locus in question [14]. Note that the scoring function for each inheritance vector is independent of any locus: a given score applies to any locus with the corresponding inheritance vector but must be weighted by the inheritance vector's probability at each locus. Performing linkage analysis therefore consists of calculating scores for each inheritance vector, visiting each locus and weighting the score of a particular inheritance vector/state by its multipoint probability at that locus. The sum of the weighted scores of each inheritance vector at a locus constitutes the LOD or Z score for that locus [14].

A trade off of Hapi's efficiency is that it does not consider all possible states at each locus. Specifically, it omits any states that exhibit double recombinations within a series of uninformative loci and may avoid considering some possible haplotype assignments at partly informative loci depending on the recombinations at surrounding loci. States that exhibit such short double recombinations are extremely unlikely according to the user-specified recombination frequencies, and are even less likely than these probabilities suggest because of recombination interference in meiosis. The low probability of such states substantially limits their potential impact on the overall linkage likelihood score at a locus. As well, if such a state did have a high linkage score, the physical limitations of double recombinations within a short span of uninformative loci would make it highly suspect. Note that Hapi does model and identify double recombinations that are supported by transmissions at informative loci. It could identify cases in which a disease locus segregates with such a double recombination. We believe that Hapi's lack of modeling double recombinations within (typically very short) spans of uninformative loci will not miss any meaningful information in standard genotype datasets. For a family with a very long span of informative loci for one or both parents, it may be fruitful to consider other algorithms. However, the results for such data will be inconclusive since the outcomes of meiosis are effectively guessed at and are unsupported by the data in this case.

Hapi does not model recombinant states at uninformative loci, but determines the maximum likelihood location of recombination between every pair of informative loci (see Section 3.2). This approach suffices in order to place an apparent recombination between the most likely pair of markers for maximum likelihood haplotype calculations. For linkage analysis, because a recombination could occur anywhere between two informative markers, we calculate the recombination frequency over the entire distance between these markers. This more accurately reflects the transition probability between informative loci. The recombination frequency is therefore

$$\phi_{lp} = \sum_{i=1}^{n} \theta_{l_i}, \tag{5.1}$$

and the algorithm can use this probability to calculate transitions between states and to compute weighted likelihood scores at informative loci.

5.2 Imputation

Researchers have recently been interested in genotype and sequence *imputation*, in which a set of individuals, assayed at a subset of genotypes of some other set of individuals, have their missing genotypes or sequence data inferred from the individuals with more data [12]. This approach relies on haplotypes being available or inferred for all the individuals.

For family-based studies, imputation using an individual that is haplotyped at a larger set of genetic data than related individuals is relatively simple. Hapi calculates inheritance vectors that designate which homolog a parent transmitted to a child. If additional haplotyped genomic data—either genotypes or sequence data—is available for a parent, we can infer that the children received the alleles for larger set of markers that reside on the homolog the child received from this parent. If additional haplotype data is available for a child or children, we can infer those portions of the parents' homologs that the child/children received. We can also deduce the larger haplotype segments that siblings received if they inherited the same homolog as the child/children with the larger marker set. At recombination sites, which occur infrequently across the length of the chromosome, the transmitted homolog is ambiguous and we cannot infer which genotypes or sequence data was transmitted.

As noted, this imputation requires that the individual(s) with the larger set of genomic data be haplotyped. If they are not, the algorithm can still infer transmissions to or from parents at locations where the sampled individual is homozygous. If both a parent and a child or multiple children have a larger set of data, we may be able to deduce their haplotypes at these additional loci using the techniques described in Chapter 6.

Chapter 6

Haplotyping Loci and Nuclear Families with Missing Parent Data

Not all nuclear family datasets contain genotype information for all family members, and parents are no exception to this. The description of our algorithm to this point has required that genotype data be available for both parents in order to haplotype a family. This requirement prevents Hapi from addressing nuclear families with genotype data for only one parent as well as families with data for siblings only and no parent data. The requirement also makes it impossible to haplotype loci that are missing data for one or both parents because of genotyping errors.

We have devised an extension to the basic Hapi algorithm that can perform haplotype inference at loci or for entire families for which genotypes for one or both parents are unavailable. Our approach to this problem is similar in nature to the approach we use in Hapi. Specifically, the algorithm determines which parent genotypes are consistent with the children's genotypes at a locus. It then builds states corresponding to the possible phase assignments for these genotypes. At the same time, the algorithm uses all applicable optimizations (Section 2.4) to reduce the number of states it must track and thereby improve efficiency.

Note that besides simply haplotyping the individuals whose genotypes are available, this algorithm also infers the most likely haplotypes of the missing parent or parents.

Note that in the discussion below, each state must be weighted by the allele frequencies of the inferred parent genotype(s).

6.1 Building Haplotype States Based on Children's Genotypes

The extension to Hapi uses the observed children's genotypes at a locus to infer all possible genotypes for the parent or parents with missing data at that locus. When both parents are missing data, there are more possible scenarios and therefore more states are necessary to perform haplotyping. Because it is more complicated, we discuss the case when we have genotypes for siblings only and no parent. Figure 6-1 gives a listing of all possible children's genotypes and the genotypes for both parents that may give rise to the children's genotypes. Additionally, the figure lists the number of states necessary to model each of these possible parent genotypes, and an overall total number of states across all possible parent genotypes that are necessary to haplotype a locus with the corresponding children's genotypes. To model an allele that a parent did not transmit to any child, the figure uses the ? symbol, signifying that the value of this allele is unknown.

This section talks about how the haplotyping algorithm works, and specifically about how this extended algorithm builds states for various scenarios of children's genotypes. Section 6.2 discusses the runtime complexity of this algorithm.

We structure the discussion below according to the number of observed alleles in the children's genotypes. The text explains Figure 6-1, outlining all possible children's genotypes and the possible parent genotypes that can produce each. The extended algorithm differs from the basic algorithm only in that the parents' genotypes are unknown, so it must build states corresponding to all possible parent genotypes that are consistent with the children's. The algorithm produces states corresponding to all possible phase assignments for each possibility of parents' genotypes, just as the basic Hapi algorithm does. This suffices in order to explore all possible haplotype assignments.

The details in Sections 6.1.1 through 6.1.4 explain why Figure 6-1 properly identify the possible parents' genotypes and tells which states the system must be built in every situation. As we explain below, the algorithm need not initially consider any states with ? alleles, but can instead consider the parent as homozygous and later determine whether the parent is likely to have transmitted only one allele.

6.1.1 Children's Genotypes with One Allele

When all the children at a locus exhibit the same homozygous genotype (e.g., a/a), the parents have four possible genotypes. Both parents may be homozygous with genotype a/a, both parents may be heterozygous with genotype $a/?$, or one or the other of the two parents may be heterozygous with genotype $a/?$ and the other homozygous as a/a.

It suffices to ignore the states with $a/?$ heterozygous genotypes. If both parents are homozygous, the algorithm cannot observe recombination. A state that does not exhibit recombination will always have higher likelihood than one that does, and consequently, the states in which both parents are homozygous will always have higher or the same likelihood as those in which one or both parents are heterozygous with genotypes $a/?$. The maximum likelihood state at this locus may include a heterozygous parent, but for this to be the case, the inheritance vector either will not exhibit recombination, or will include a recombination that the algorithm observes at an adjacent locus.

In an analogous fashion to the handling of uninformative loci in Hapi, this extended algorithm initially skips loci in which children exhibit only one allele. Subsequently, when the maximum likelihood inheritance vectors at upstream and downstream loci are known, the algorithm can determine the most likely location of recombinations between those two loci. A recombination can occur at a locus of the type we are discussing (in which the children's genotypes contain only one allele), but this will only occur if the event is supported by surrounding loci.

With the inheritance vector at this type of locus known, the algorithm determines whether either parent did not transmit one of its homologs. If so, the allele value of the untransmitted homolog is unknown and the algorithm assigns a ? allele to that homolog.

Note that because the children are homozygous, it is not necessary to phase them. The algorithm therefore deduces as much as possible about the parent's genotypes at this locus based on the children's genotypes and the maximum likelihood inheritance vectors at surrounding loci. Because the parents only transmit an a allele, the only information this process can deduce is whether the parent is likely to be homozygous or not.

6.1.2 Children's Genotypes with Two Alleles

We can glean more information from loci at which the children exhibit two or more alleles. Note that most SNPs are bi-allelic and consequently we will typically only encounter loci with one or two alleles in datasets containing SNP genotypes.

If the children's genotypes consist only of a/a and b/b, then both parents must have the genotype a/b. In this case, though the parents share the same heterozygous genotype, all the children are homozygous and therefore there are only four possible inheritance vectors, analogous to a fully informative for both parents locus. Because of equivalent inheritance vectors, such a locus produces only one state and back tracing can resolve the haplotypes at any upstream loci.

For a locus with children's genotypes a/a and a/b, there are effectively three possible sets of parents' genotypes. One set includes one heterozygous parent with genotype a/b while the other parent is homozygous with genotype a/a. The second set assigns these heterozygous and homozygous genotypes to opposite parents. The last possible set of genotypes assigns both parents the heterozygous genotype a/b. Note that the parent that is assigned a homozygous genotype may not have transmitted one of its homologs, and in such a case, the genotype for that parent is $a/?$. The algorithm handles the two cases where each parent is homozygous similarly to the case where all children display only one allele. It leaves the parent's genotype as homozygous initially and later, after determining the maximum likelihood inheritance vector at the locus, it assigns a ? allele to the homozygous parent if it transmitted only one allele.

As in the original algorithm, this extension to Hapi maps all the states at the previous locus to the states at the current locus. If the same inheritance vector value results from different parent genotypes, it suffices, as an application of dynamic programming, to only retain the parent genotype assignments that yield maximum likelihood (or minimum recombinations). All downstream loci will exhibit the same recombinations for a given inheritance value regardless of which parent genotypes appear at the current locus.

If the children at a locus exhibit genotypes a/a, b/b, and a/b, the only consistent genotype assignment for the parents is for both to be a/b. This is the same as partly informative locus in the original algorithm.

The last possible way children can exhibit two alleles at a locus is for all to have genotypes a/b. Here, there are three possible sets of parent genotypes. One set has one parent homozygous with genotype a/a and the other homozygous as b/b. Another set uses the same genotypes assigned to opposite parents. The last set assigns both parents the heterozygous genotype a/b. The first two cases yield no information about meiosis and it is impossible to determine which parent transmitted each of the two alleles. Therefore, even if we knew that both parents are homozygous, the algorithm cannot determine the parents genotype values nor the phase of the children. If the parents both have genotypes a/b and all children are heterozygous with this same genotype, the case is again ambiguous. Even if a particular inheritance vector value is known to apply in this case, there are two phase assignments for the parents that will be consistent with this inheritance vector.

The case where all children have genotypes a/b is almost entirely ambiguous and the extended algorithm therefore omits analyzing these loci for the purpose of haplotyping. Once the algorithm determines the inheritance vectors of the surrounding loci, it can evaluate whether both parents being heterozygous would fit the data well. If any assignment of parent alleles to homologs for these heterozygous genotypes produces an inheritance vector that is the same as the upstream or downstream inheritance vector (or an intermediate value between the two), then it is possible that the parents are indeed heterozygous. Usually, when the children all have the same a/b genotype, the parents genotypes will be homozygous, particularly if the number of children in the family is large. (For a large family, if the parents both have genotypes a/b, we would expect at least one child to receive a homozygous genotype.) This evaluation can assure us that the parents are not heterozygous, but still cannot resolve which parent is homozygous for each of the alleles and therefore cannot determine the children's phase. It is therefore sensible to mark these loci as ambiguous.

6.1.3 Children's Genotypes with Three Alleles

When children exhibit three or four alleles, the analysis is much simpler. We let the reader refer to the figure for the possible children's genotypes and the parents' genotypes that are consistent with these.

There are two classes of possible children's genotypes with three alleles. The first class yields only one consistent set of genotypes for the parents. Since the parents' genotypes are heterozygous but with differing alleles in this case, the locus is fully informative for both parents and the algorithm retains only one state.

For the second class listed in the figure where the children exhibit genotypes a/b and a/c, there are two consistent sets of genotypes for the parents. Both parents may contain an a allele, and in this case, both parents are heterozygous with differing genotypes. Thus, for one set of parents' genotypes, the algorithm retains only one state. Alternatively, only one parent may contain an a allele while the other parent has genotype b/c. For this case, the parent with the a allele could be either homozygous or heterozygous but only transmit the a allele. As with previous genotype values

# of Unique Children's Alleles	Genotypes			For n Previous States	
	Children's	Parent p	Parent q	# of States	Total States
1 Allele	a/a	a/a	a/a	–	–
		$a/?$	a/a		
		a/a	$a/?$		
		$a/?$	$a/?$		
2 Alleles	a/a, b/b	a/b	a/b	1	1
	a/a, a/b	a/b	a/a $a/?$	$\leq n$	$\leq 6n$
		a/a $a/?$	a/b	$\leq n$	
		a/b	a/b	$\leq 4n$	
	a/a, b/b, a/b	a/b	a/b	$\leq 4n$	$\leq 4n$
	a/b	Ambiguous			
3 Alleles	a/a, a/b, a/c a/a, a/b, b/c a/a, a/b, a/c, b/c a/b, a/c, b/c	a/b	a/c	1	1
	a/b, a/c	a/c	a/b	1	$\leq n+1$
		a/a $a/?$	b/c	$\leq n$	
4 Alleles	a/b, c/d	a/c	b/d	1	2
		a/d	b/c	1	
	a/b, c/d, a/d a/b, c/d, b/c a/b, c/d, a/d, b/c	a/c	b/d	1	1
	a/b, c/d, a/c a/b, c/d, b/d a/b, c/d, a/c, b/d	a/d	b/c	1	1

Figure 6-1: Possible parent genotypes that may give rise to the given children's genotypes. The listing is organized by the number of alleles the children exhibit. Our extended algorithm for haplotyping families with no genotype data for the parents uses this information to determine which parent genotypes it must build states for. For each set of parent genotypes, the table shows the number of states needed to represent the possible parent phase assignments and corresponding inheritance vector values. The figure also gives the total number of states for each of the possible children's genotypes.

of this form, the algorithm initially assigns the genotype as homozygous and later evaluates whether the children received alleles from both homologs of the parent.

6.1.4 Children's Genotypes with Four Alleles

Whenever the children exhibit four alleles, the parents must have heterozygous genotypes with differing alleles. In only one case does the algorithm retain more than one state. If the children exhibit genotypes a/b and c/d, the data do not indicate which parent has each of the alleles. The parents cannot have the same genotypes as the children since the same parent cannot transmit the a allele and the b allele (and likewise for the c and d alleles). Therefore, two sets of parents' genotypes are possible, as the figure shows, and the algorithm builds states corresponding to each. Ultimately only two states result at this type of locus, and a downstream locus can resolve which of these is more likely.

Whenever children exhibit at least three different genotypes with four alleles, the data are sufficient to deduce what the parent genotypes must be. The figure shows the possible cases, and for the two sets of parents' genotypes, the algorithm retains only one state.

6.1.5 Initial State

In a similar fashion to the way Hapi builds initial states for full-data families, this new algorithm processes loci and builds an initial state as it encounters them. When the children's genotypes contain three or more alleles, a locus that results in only one state will occur early in the analysis of a chromosome and haplotyping can proceed from that point.

For datasets containing only bi-allelic genotypes, including most SNP datasets, the first states the algorithm builds may have undefined inheritance values for the meioses from parents that are homozygous. For example, if the children's genotypes at the first locus are a/a and a/b, one of the parents may be homozygous. The system assigns the inheritance vector values corresponding to transmissions from a homozygous parent as undefined when there is no previous locus state. It then propagates these undefined inheritance values forward until it eventually encounters a locus for which the only compatible parent genotypes are both heterozygous.

When no genotype data exist across all loci for both parents, if the first locus the system encounters has children with genotypes a/a and a/b, the algorithm will arbitrarily choose one parent to assign as heterozygous and the other as homozygous. (Note that the system also builds states at this locus in which both parents are heterozygous.) In this case of neither parent having genotype data, the algorithm cannot distinguish between the two parents. If it were to build states with homozygous genotypes for each of the parents, they would result in identical likelihoods for the same haplotypes assigned to opposite parents. It suffices to arbitrarily choose one parent as possibly being homozygous at this locus. Later, at downstream loci, regardless of the kinds of states built at earlier loci, the algorithm must build states corresponding to each of the parents being homozygous. The first locus to have a state built will have defined differences between the two parents so the algorithm must now search both possibilities of homozygous parents.

To build initial states corresponding to partly informative parent genotypes (i.e., both parents having a/b genotype), the system marks any heterozygous children's inheritance vector values as ambiguous. This is necessary since it is impossible to determine, until some later locus, which parent transmitted each of the alleles. We therefore represent the possibility of either value in the form of an ambiguous inheritance value. The system finds the actual state of the inheritance downstream at another locus, and in particular, at a locus that is partly informative but where the child is homozygous. Note that if the parents are heterozygous with the same genotype, a child has a $\frac{1}{2}$ chance of being heterozygous and a $\frac{1}{2}$ chance of being homozygous. Thus we are very likely to encounter a downstream locus that resolves ambiguous inheritance vector values.

6.2 Runtime Complexity

The runtime complexity for haplotyping loci and families with missing parent data is likely to be slower than that for haplotyping families with data for both parents. However, we expect that the runtime should be reasonable, especially for families that have data for one parent.

A bi-allelic SNP genotype dataset for a family with no genotype data for either parent will pose the biggest computational burden to our algorithm. Since the algorithm initially omits processing loci for which the children are all homozygous with the same genotype, we consider here the cases in which the children's genotypes exhibit two alleles.

If the children have genotypes a/a, b/b, and a/b at a locus, the locus must be what we have termed partly informative. In general, for n states at the previous locus, the algorithm may produce $4n$ states at a partly informative locus. It would be unacceptable for the number of states to quadruple at each new locus, but this is merely an upper bound. If all children were homozygous, only four states would result, so the state expansion comes as a result of heterozygous children. Constraints imposed on the inheritance values at previous loci make it likely that the number of states will stay to a reasonable number. As we explained in Section 4.3.2, any series of partly informative loci has a polynomial bound on the number of states that can result.

A more complex situation can result because of the interplay between the loci that have a possible fully informative for one parent type locus and the partly informative type of loci. As we have explained elsewhere (Section 4.3.2), these two locus types can produce an exponential blowup when intermixed. The requirements for this to occur are extremely strict. The most unlikely of the requirements for an exponential blowup is that children must stay heterozygous at so-called partly informative loci. As previously mentioned, a child has a $\frac{1}{2}$ chance of being heterozygous and a $\frac{1}{2}$ chance of being homozygous. Consequently, the number of states cannot grow large over a extended number of loci.

The fact that homozygous children have a limited set of possible inheritance vector values, regardless of the parents genotypes, makes the approach we have outlined feasible. Although the algorithm's runtime for families without genotype data for either parent may be markedly less than that for families with data for both parents, the optimizations we employ will improve efficiency compared with other algorithms. Specifically, the ability to collapse multiple states into one state through the use of ambiguous genotypes is unique to our approach and results in improved efficiency over existing work.

This algorithm should handle families with data for one parent relatively efficiently, as the known genotype data for that one parent reduces the number of partly informative loci the algorithm must check.

6.3 Memory Usage

The overall memory usage for our approach to haplotyping families and loci with missing parent data is higher than for the algorithm that applies to parents with full data. Even so, the memory requirements for the full-data algorithm is extremely low and the overhead for extending the algorithm to this case of missing parent data is not excessive. Hapi's optimizations guarantee that its asymptotic memory requirements are less than or equal to those for other algorithms.

At any point the haplotyping process, the algorithm can remove any states that are unreachable from the states at the current locus. The inheritance vector values for homozygous children at a partly informative locus have only four possible values. A child has a $\frac{1}{2}$ chance of being homozygous at such a locus, and the genotypes for children with identical or opposite inheritance values follow a pattern: all be either homozygous or heterozygous at a given locus. Often two loci that the algorithm detects as partly informative will appear in succession with all children being homozygous at one of the two loci. The two successive partly informative loci that do not exhibit recombination have a $\frac{1}{2}$ chance of this occurring. The homozygous genotypes of the children that occur at these loci will limit the number of states that can occur at the second locus to only twelve. This situation is a reasonable time for the algorithm to perform back tracing to free the memory associated with any unreachable states.

Note that loci that are fully informative for one or both parents are harder to detect when we do not have access to genotype data for both parents. If genotype data is available for one parent and that parent is homozygous at a locus, any heterozygous children indicate that the parent with missing data is heterozygous and that the locus is therefore fully informative for that parent. When such a locus occurs subsequent to a one that is fully informative for the parent for which we have genotypes, the algorithm can eliminate all but the most likely (or minimum recombinant) haplotype state and perform back and forward tracing to assign the haplotype values at preceding loci.

Chapter 7

Haplotyping Multi-Generational Pedigrees

The ability to haplotype moderate to large sized multi-generational pedigrees is a significant outstanding problem that no program yet addresses. Existing techniques, limited by both time and space requirements, cannot handle pedigrees of more than roughly 20 non-founders. Non-founders are individuals with at least one parent in the pedigree. For non-founders, we can observe the results of meiosis and thus they have two bits in the inheritance vectors at each locus.

Current algorithms for haplotyping pedigree datasets have exponential complexity in general. GENEHUNTER [14] first introduced the optimization that avoids representing equivalent inheritance vectors for non-founders. Using this optimization, a pedigree with n non-founders and f founders has 2^{2n-f} possible inheritance vectors at each locus rather than 2^n. (Hapi applies this optimization to nuclear families where $f = 2$ for the two parents and thereby reduces the state space by a factor of $2^2 = 4$.) Other optimizations exist, such as avoiding states that are inconsistent with the genotypes at a locus and identifying state space redundancies [22, 1], but even with these optimizations, existing algorithms do not scale.

At present, researchers that work on large pedigrees can only haplotype such data by breaking them apart into small portions for analysis. Breaking up the pedigree produces non-optimal results since the information from one part of the pedigree influences the relative likelihoods of different haplotype assignments. The probability of these assignments may be nearly identical when considered separate from the larger pedigree. The pedigree may also rule out a haplotype assignment that otherwise appears to have high likelihood.

We present here an algorithm that applies and adapts the techniques implemented in Hapi to haplotype multi-generational pedigrees. This algorithm can haplotype much larger pedigrees than current techniques can address. We first describe why one approach—to build states completely separately for each nuclear family and, when back tracing, to choose the maximum likelihood states from each family—fails. We then present how to apply and expand on Hapi's optimizations to handle pedigrees.

Throughout this chapter we will refer to *shared individuals* as pedigree members who appear as a child in one nuclear family and a parent in another.

This chapter refers only to the maximum likelihood problem for pedigrees, but the minimum recombinant problem works analogously. Instead of maximizing probabilities, the minimum recombinant algorithm uses integer counts and attempts to minimize these across all loci.

7.1 Pitfalls of Building Separate Haplotype States for Each Nuclear Family

A natural way to think about extending Hapi to multi-generational pedigrees is to analyze each nuclear family separately and consider a pedigree state as the composition of states sampled from each family. In fact, it is the case that we can decouple the probabilities of the inheritance values across nuclear families, but only *at a single locus*. This approach ignores states at surrounding loci.

It is straight forward to find, at a given locus, the composition of states across families that has consistent phase assignments for shared individuals and produces maximum likelihood. However, each of these states contains pointers to states at previous loci that affect this maximum likelihood probability. The maximum likelihood phase assignment for a shared individual at some previous state in one family may differ from the phase assignment for that individual in another family. As such, the composition of states may yield infeasible assignments. The algorithm cannot arbitrarily change to another state during back tracing since the original state probability accounts only for one path of states. Changing paths at some upstream locus may yield a path of states with lower probability than some other path that starts at the first (downstream-most) locus.

A similarly unworkable approach to this problem is to calculate for each family and locus the maximum probabilities for the two possible phases of each shared individual. The algorithm would store both the maximum probabilities and the corresponding inheritance vectors for each family at each locus. The false intuition in this dynamic programming algorithm is that the system could choose between the two phase assignments at each locus in order to resolve differences between the families of the shared individual. This approach suffers from the same problem as the first. The probabilities of the inheritance vector values at a particular locus depend on the assignments at other loci, so arbitrarily choosing at each locus creates a series of inheritance vectors whose probability is unknown.

In general, dynamic programming works by representing only the optimal path to a given state, evaluated according to some objective function such as maximum likelihood. The results do not include any non-optimal path, and the algorithm can only evaluate the probabilities of non-optimal paths by explicitly considering them. Dynamic programming algorithms are unaffected by which path yields the optimal result. To obtain the results of the objective function for all paths requires computing the probability across each such path, and there are an exponential number of such paths. For haplotyping, a slightly suboptimal path of states for one nuclear family may actually have maximum likelihood in the context of the entire pedigree.

The algorithm must therefore account for the probabilities of haplotype assignments evaluated over the entire pedigree and not individual nuclear families.

7.2 Using Hapi's Optimizations to Build Pedigree States

In light of the need to evaluate the haplotype dependence between pedigree families not just at a single locus, but across all loci, it is necessary to build states that represent the entire inheritance vector for all pedigree members. Each such state stores the maximum probability of that inheritance vector, directly incorporating the probabilities of states at the previous locus that reach it. As with the normal Hapi algorithm, states keep pointers to the previous locus state or states that yield the overall maximum likelihood. This formulation implicitly encodes the dependence of the probability of a haplotype assignment on the need for a consistent phase for shared individuals across loci. The algorithm does not build any states that have inconsistent haplotype assignments for any shared individuals since these have 0 probability.

Existing algorithms consider entire pedigree inheritance vectors and perform calculations relative to them. Our algorithm necessarily builds states that are full inheritance vectors, but we perform calculations and optimizations in the context of each nuclear family in the pedigree. Throughout this chapter, we use the term *inheritance segment* to refer to the partial inheritance vectors that apply to each nuclear family.

In order to correctly build full inheritance vectors and states based on per-family inheritance segments, three invariants must hold. First, each shared individual must have the same phase for each family it is a member of. Second, we must build inheritance vectors corresponding to all possible combinations of each family's inheritance segments. Finally, we must consider all states at the previous locus that can transition to the newly built state at the current locus. We explain below how our algorithm meets each of these conditions.

At each locus, the algorithm first computes the possible inheritance segments for each nuclear family in the pedigree. It does this the same way Hapi does, considering all possible phase assignments for the parents and using the inheritance segments that occur for that family at the previous. As it builds these inheritance segments, the algorithm calculates and stores the local probability of transitioning from the inheritance segments that occur for this family at the previous locus to the current one. This saves the work in calculating transition probabilities for all inheritance vectors since each inheritance segment for a family will appear in multiple inheritance vectors.

Figure 7-1 shows example inheritance segments for two families and associated probability values for transitioning from a given inheritance segment at the previous locus to one at the current locus. We use α values to designate transition probabilities for family 1, β values for transition probabilities for family 2. α_{11} is the probability of transitioning from the previous inheritance segment designated as 1, i.e., `0101`, to the current inheritance segment designated as 1, i.e., `0110`. The value α_{21} is the

	Inheritance Segment		Probability
	Previous	Current	
Family 1	0101	0110	α_{11}
	0101	1001	α_{12}
	⋮	⋮	⋮
	1101	0110	α_{21}
	⋮	⋮	⋮
Family 2	111010	101110	β_{11}
	111010	111011	β_{12}
	⋮	⋮	⋮
⋮	⋮	⋮	⋮

Figure 7-1: The pedigree haplotyping algorithm computes *inheritance segments* for each nuclear family. A pedigree inheritance vector is formed by concatenating an inheritance segment from each nuclear family. The algorithm calculates and stores the local transition probabilities from the inheritance segments for a family that occur at the previous locus to the those at the current locus. We use different symbols to encode the probabilities for inheritance vectors for each family, with subscripts to indicate which previous and which current inheritance segments the probability corresponds to.

probability of transitioning from the previous inheritance segment labeled 2, which is 1101, to the current inheritance segment labeled 1.

In order to build full pedigree states/inheritance vectors, the system concatenates inheritance segments from each family. To keep inheritance vectors consistent across states, the algorithm concatenates the inheritance segments from the nuclear families in the same order at each locus. The order of concatenation has no effect on the algorithm's performance.

Figure 7-2 shows concatenated inheritance vectors and their probabilities using the example inheritance segments and transition probabilities from Figure 7-1. The probability of an inheritance vector is the maximum probability across all previous states that may map to this vector value. A probability for one such mapping is the product of the transition probabilities for the nuclear families' inheritance segments multiplied by the overall probability of the previous state. In Figure 7-2, we use P_x to denote the probability of some previous state x.

The algorithm does not form any inheritance values with inconsistent phase assignments for shared individuals. Rather than explicitly checking each possible pairing of inheritance segments involving shared individuals, the algorithm classifies each inheritance segment according to its phase for any shared individual(s). For families in which a parent is a shared individual, the algorithm keeps two lists identifying the inheritance segments that correspond to both phases. For every inheritance segment in which the shared individual is a child, the algorithm forms inheritance vectors by pairing that segment with each segment from the list that has the same phase.

Inheritance Vector	Probability
$\underbrace{\texttt{0110}}\,\underbrace{\texttt{101110}}\cdots$	$\max\{\alpha_{11} \cdot \beta_{21} \cdots P_a;$ $\alpha_{21} \cdot \beta_{21} \cdots P_b;$ $\cdots\}$
$\texttt{0110}\,\texttt{111011}\cdots$	$\max\{\alpha_{11} \cdot \beta_{22} \cdots P_c;$ $\alpha_{21} \cdot \beta_{22} \cdots P_d;$ $\cdots\}$
$\texttt{0110}\quad\cdots$	$\max\{\alpha_{11} \cdots; \alpha_{21} \cdots;$ $\cdots\}$
$\vdots$	$\vdots$
$\texttt{1001}\,\texttt{101110}\cdots$	$\max\{\alpha_{12} \cdot \beta_{21} \cdots P_e;$ $\cdots\}$
$\vdots$	$\vdots$

Figure 7-2: The algorithm forms inheritance vectors by concatenating an inheritance segment from each nuclear family, with segments from each family appearing in the same position across vectors. The probability for a particular inheritance vector is the maximum across all previous states that may reach it multiplied by the product of the transition probabilities for transitioning across inheritance segments.

At many loci, we can unambiguously determine a shared individual's phase based on its genotype and the genotypes of its parents. Before even forming any inheritance segments for the nuclear families, the algorithm inspects the genotypes of the shared individuals and their parents. Wherever possible, it deduces and assigns each shared individual's phase. Subsequently, when forming the inheritance segments for the nuclear family in which the shared individual is a parent, the algorithm only builds inheritance segments corresponding to the individual's phase, if it was assigned. This enables the system to avoid ever building any 0 probability inheritance segments for the shared individuals' children.

To build an inheritance vector/state at the current locus, the algorithm considers each state at the current locus. It then uses the possible transitions to inheritance segments for each nuclear family to build all possible combinations of inheritance vectors it may map to. This process utilizes the phase constraints for shared individuals mentioned above to avoid considering any combinations of inheritance segments that are impossible. The probability of transitioning from the previous state under consideration to each state the algorithm builds is the product of the individual inheritance segment probabilities multiplied by the previous state's probability. This process ensures that we consider all states at the previous locus and the states that they can possible map to at the current locus.

We now describe how the optimizations present in Hapi significantly reduce the number of states that the algorithm must consider in order to haplotype pedigrees.

7.2.1 Ambiguous Inheritance Values

The use of ambiguous inheritance values dramatically reduces the number of states that Hapi represents (see Section 2.4.2). This same optimization can apply to pedigrees, and specifically to all non-shared individuals. For non-shared individuals, ambiguous inheritance values work the same as in haplotyping nuclear families. Note that all pedigrees include some non-shared individuals: any individual without children in the pedigree are necessarily a member of only one nuclear family. Thus, the last generation in a pedigree contains all non-shared individuals. Note that most pedigrees contain a large proportion of non-shared individuals.

For shared individuals, the system must use unambiguous inheritance values since ambiguous inheritance values simultaneously represent two phase assignments. All shared individuals must have consistent, known phase in each nuclear family they reside in, and therefore they cannot have an ambiguous inheritance value.

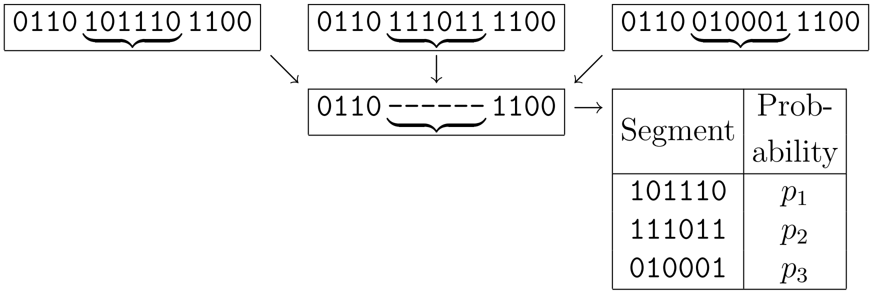
If a nuclear family has only one child that is a shared individual, the number of inheritance segments for that family only doubles compared to the number needed to haplotype the nuclear family alone. However, the system must represent all combinations of unambiguous values for all shared individual children in each family. Thus, if a family has i shared individuals, the number of states for that family can increase by a factor of up to 2^i. Note however that only heterozygous individuals at a partly informative locus can take on the two inheritance values corresponding to ambiguous inheritance. Thus, even if a nuclear family contains multiple shared individual children, the number of states actually increases exponentially in the number of *heterozygous* shared individuals. As well, as we earlier mentioned, this is an increase in the number of inheritance segments relative to the application of Hapi's optimizations in the context of a nuclear family. The algorithm's use of ambiguous inheritance values for non-shared individuals will always reduce the number of necessary states relative to other algorithms.

7.2.2 Uninformative Loci

Hapi skips processing uninformative loci for nuclear families since they provide no information about meiosis. In our extended algorithm for multi-generational pedigrees, we cannot skip these loci entirely because usually one of the nuclear families will be informative at the locus. If all the nuclear families in the pedigree are uninformative at a locus, then the algorithm can skip that locus, but we don't expect that this will occur often.

The algorithm accounts for recombination probabilities in a family only at informative loci and does not consider recombinant inheritance segments for any nuclear families that are uninformative. Consequently, when building the pedigree inheritance vectors, the algorithm copies exactly each inheritance segment that occurs at the previous locus for all uninformative families. It does not need to calculate any recombination probability for transitioning from the previous locus for these families, so less work is required for them.

It may seem possible to omit the inheritance segments corresponding to unin-



Segment	Probability
101110	p_1
111011	p_2
010001	p_3

Figure 7-3: Collapsing inheritance vectors at loci where a family has an uninformative locus is equivalent to building full inheritance vectors. The algorithm must store the probabilities associated with each inheritance segment in each collapsed state, and subsequent loci cannot keep pointers to a previous state's values since different previous collapsed states will yield different likelihoods for each inheritance segment.

formative nuclear families and temporarily "collapse" the inheritance vector values. Figure 7-3 illustrates this idea. The algorithm would store a list of possible inheritance segments for the uninformative family and later form complete inheritance vectors at a locus where that family is informative based on this list. Unfortunately, this fails since the previous locus states that map to the collapsed state will have different probabilities depending on the inheritance segment for the uninformative nuclear family. Each state must therefore keep track of the probability associated with each inheritance segment and could do so in a table, as Figure 7-3 shows. This is equivalent to (and requires more bookkeeping than) building the full inheritance vectors.

One possible extension of this idea that would seem to improve efficiency is to share tables across loci for collapsed states. In this approach, a collapsed state keeps a pointer to the table for the previous state that yields maximum likelihood at the current locus. The problem with this is that the probabilities of inheritance segments will differ between collapsed states. One state may yield higher likelihood for a particular inheritance segment while another state produces higher likelihood for a different segment. The algorithm would therefore have to build a new table at each new collapsed state at a locus which, as we earlier mentioned, is equivalent to keeping full pedigree states. Our extended algorithm therefore builds full inheritance vectors, including the possible inheritance segments for any uninformative families.

7.2.3 Fully Informative for One Parent Loci

Nuclear families with loci where one parent is homozygous and the other is heterozygous are similar to families with uninformative loci. The algorithm need not consider any inheritance segments in these families that exhibit recombination from the homozygous parent. Just as Hapi does, this extended algorithm includes the probability of recombination at the next locus in which the now homozygous individual is heterozygous.

Note that when a shared individual is homozygous, the inheritance segments in each of that shared individual's families do not affect each other. Similar to the uninformative loci, it is tempting to separate the inheritance vectors into two parts: one corresponding to the sub-pedigree that includes the shared individual as a child, and another for the sub-pedigree in which the shared individual is a parent. This again fails; the probabilities of states at the previous locus vary and these inheritance vector parts have different probabilities depending on which previous state maps to them.

It is worth discussing one way of attempting to use sub-pedigree inheritance vectors and why it does not work. If there are a series of loci in which a shared individual is homozygous, the algorithm can compute the maximum likelihood path across these loci for the states of each sub-pedigree. To do this, it sets the probability of all sub-pedigree states to 1 at the first locus that is homozygous. The intuition is that the algorithm can calculate the maximum likelihood path across this series of loci *relative to* the unknown probability of this initial states.

Unfortunately, it is incorrect to assign the same probability to all sub-pedigree states at the first homozygous locus. The previous locus will not map to all states with the same probability. For example, the path from the previous locus to one of these first locus states may have very high probability, and the path to another state, very low probability. Yet, the algorithm, ignoring variation in the probability of reaching these states, may produce a "maximum likelihood" path that includes the low probability state. Across this short series of loci, the low probability state may not exhibit any recombination in the members of the sub-pedigree, or it may recombine less than the other, higher-probability state. The result is that the algorithm locates a path that is less likely across the entire chromosome length, even though it has higher likelihood within this short span of loci.

In light of these issues, our algorithm includes full inheritance vector values in states that have families with fully informative for one parent loci. Note that our approach of never introducing recombinations for transmissions from a homozygous parent represents a significant efficiency gain over existing algorithms. These other algorithms must explore most or all possible homolog transmissions from homozygous parents, including very unlikely scenarios that include many recombinations.

7.2.4 Equivalent States

What we have termed the equivalent states optimization was original discovered by Kruglyak et al. and applies to all founders in a pedigree [14]. Our extended algorithm leverages this optimization in addition to those discussed above. This optimization reduces the necessary states by a factor of 2 for each founder. Thus, the number of states is reduced by a factor of 2^f, where f is the number of non-founders.

7.3 Missing Data

If a shared individual is missing data, the algorithm uses the techniques discussed in Chapter 6 to determine the possible genotypes for the individual according to the genotypes of that person's children. As well, it utilizes the genotypes of the shared individual's parents, where available, to rule out any impossible genotypes. For example, if one of the shared individual's parents has genotype b/b, the shared individual cannot have genotype a/a.

Otherwise, for any non-shared parents or children, the algorithm relies, respectively, on the techniques described in Chapter 6 and Section 2.8 to perform haplotyping.

7.4 Complexity Analysis

Suppose a pedigree has r nuclear families and that there are $s_1, \ldots, s_r$ inheritance segments for each of these families at some locus. Then this locus has $S < \prod_{i=1}^{r} s_i$ states; the total number of states will be less than the product on the right-hand-side because some state combinations have inconsistent phase for shared individuals and because of the use of optimizations. Typically half of the state combinations for each heterozygous shared individual will be inconsistent across families.

Let L be the number of marker loci and σ be the maximum number of states that the basic Hapi algorithm would produce at any locus to haplotype all of the pedigree's nuclear families. Note that in general σ may be exponential in the number of children in the largest nuclear family, but our experimental results in Chapter 4 show that σ remains small in practice. On average, Hapi produces fewer than 7 states at the most complex locus type in order to haplotype a nuclear family (see Figure 2-3). Let i^* be the maximum number of shared individuals that occur as children in any one nuclear family. Shared individuals may increase the number of states that the extended algorithm must produce compared to the basic algorithm by a factor of 2^{i^*}. Our algorithm for pedigrees therefore scales as $O(L \cdot \sigma^{ri^*})$. This exponential factor, ri^* or the number of nuclear families times the maximum number of shared individual children in any family compares favorably to the bound for other algorithms of 2^{2n-f}. Note that this is a conservative estimate and we expect that Hapi will scale well to pedigrees that contain large numbers of nuclear families and moderate numbers of shared individuals.

7.5 Memory Usage

The algorithm outlined above does not produce a single state at any locus. This means that the algorithm cannot reclaim memory the way that Hapi does. One way the algorithm can reduce memory is to utilize a modified form of the back tracing optimization implemented in Hapi in order to reclaim memory. Whenever a family encounters a fully informative for both parent locus or two successive loci that are fully informative for alternate parents, the algorithm can eliminate any previous states that are unreachable from these states. Any haplotype assignment must pass through these states and therefore unreachable assignments cannot have maximum likelihood. In general, the algorithm can always remove states that are unreachable, but after reducing the number of states that a given family produces to a small number (in this case to four or fewer if one of the parents is a founder) is a logical time to perform back tracing and eliminate states. This will keep memory usage to a tractable size throughout computation and prevent memory usage from increasing without ever being reclaimed as the algorithm processes loci.

Back tracing to reclaim memory need only proceed to the last locus for the given nuclear family at which back tracing was last completed. This is analogous to the process in Hapi and requires very little computational effort to complete.

To avoid double freeing states that were already reclaimed by back tracing for another nuclear family, the algorithm keeps a per-locus bit field that indicates when a state has been freed.

Chapter 8

Conclusions

Assignment of haplotypes is an important element in a number of significant areas of genetic analysis, including locating genes involved in human disease, analyzing the products of meiosis to locate recombination hotspots and gene conversions, and studying population dynamics and history for humans and other species. Because of their importance, researchers have developed computational algorithms for inferring haplotypes from genotypes. The most effective approach to this problem is to use data for individuals whose family relationships are known.

Inferring minimum-recombinant haplotypes for the individuals in a pedigree is known to be NP-hard in general [5, 17]. Problems classified as NP-hard are not known to have a polynomial time (i.e., efficient) solution, and are therefore thought to be computationally intractable. Existing algorithms computing either maximum likelihood (based on recombination rates) or minimum-recombinant solutions for pedigrees consequently have exponential complexity.

Hapi is an efficient algorithm for inferring both minimum-recombinant and maximum likelihood haplotypes for nuclear families. Hapi runs in polynomial time in practice (see Appendix for algorithm complexity details), and our experimental data demonstrate the effectiveness of our approach. When haplotyping a large dataset of nuclear families, Hapi outperforms the state-of-the-art system Merlin with a speedup of between 3.8–320 times. Hapi also runs between 6.4–2460 times faster than Allegro and 17–448 times faster than Superlink.

Four key insights (implemented as optimizations) underly Hapi's efficiency. The first is that it is only necessary to build states that are consistent with the Mendelian laws of inheritance applied to the individuals' observed genotypes. Second, when a parent is homozygous, it is unnecessary to build states that represent recombinations, and our maximum likelihood approach uses a novel calculation of recombination rates to make this possible. Third, many of the possible states at a locus have equivalent inheritance vectors, and among these, only the state with the fewest recombinations or the maximum likelihood needs to be retained while the others can be eliminated. Fourth, we have formulated a novel representation of inheritance vectors that includes ambiguous inheritance values. This representation reduces the need to represent an exponential number of states at loci of the type we call partly informative.

As time passes and technology improves, genotype datasets will continue to grow in size, both numbers of individuals and numbers of loci assayed. As such, faster tools for haplotype analysis will be essential. Existing algorithms for haplotyping related individuals have hard limits on the size of families they can analyze because of their exponential complexity. These algorithms are consequently ineffective for datasets with thousands of families or for families with large numbers of children. Hapi provides a solution that is able to meet many of these future challenges.

Bibliography

[1] Gonçalo R. Abecasis, Stacey S. Cherny, William O. Cookson, and Lon R. Cardon. Merlin—rapid analysis of dense genetic maps using sparse gene flow trees. *Nature Genetics*, 30:97–101, January 2002.

[2] Brian L. Browning and Sharon R. Browning. A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *The American Journal of Human Genetics*, 84:210–223, February 2009.

[3] Sharon R. Browning and Brian L. Browning. Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *The American Journal of Human Genetics*, 81:1084–1097, November 2007.

[4] Graham Coop, Xiaoquan Wen, Carole Ober, Jonathan K. Pritchard, and Molly Przeworski. High-resolution mapping of crossovers reveals extensive variation in fine-scale recombination patterns among humans. *Science*, 319:1395–1398, March 2008.

[5] Koichiro Doi, Jing Li, and Tao Jiang. Minimum recombinant haplotype configuration on tree pedigrees. In *3rd Annual Workshop on Algorithms in Bioinformatics (WABI'03)*, pages 339–353, September 2003.

[6] Ma'ayan Fishelson, Nickolay Dovgolevsky, and Dan Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 59:41–60, 2005.

[7] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10:149–169, 1997.

[8] Guimin Gao, David B. Allison, and Ina Hoeschele. Haplotyping methods for pedigrees. *Human Heredity*, 67:248–266, 2009.

[9] Javier Gayán, Denise Brocklebank, J. Michael Andresen, Gorka Alkorta-Aranburu, The US-Venezuela Collaborative Research Group, M. Zameel Cader, Simone A. Roberts, Stacey S. Cherny, Nancy S. Wexler, Lon R. Cardon, and David E. Housman. Genomewide linkage scan reveals novel loci modifying age of onset of Huntington's disease in the Venezuelan HD kindreds. *Genetic Epidemiology*, 32:445–453, July 2008.

[10] Daniel F. Gudbjartsson, Kristjan Jonasson, Michael L. Frigge, and Augustine Kong. Allegro, a new computer program for multipoint linkage analysis. *Nature Genetics*, 25:12–13, May 2000.

[11] Daniel F. Gudbjartsson, Thorvaldur Thorvaldsson, Augustine Kong, Gunnar Gunnarsson, and Anna Ingolfsdottir. Allegro version 2. *Nature Genetics*, 37:1015–1016, October 2005.

[12] Bryan N. Howie, Peter Donnelly, and Jonathan Marchini. A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. *PLoS Genetics*, 5, 6 2009.

[13] Augustine Kong, Daniel F. Gudbjartsson, Jesus Sainz, Gudrun M. Jonsdottir, Sigurjon A. Gudjonsson, Bjorgvin Richardsson, Sigrun Sigurdardottir, John Barnard, Bjorn Hallbeck, Gisli Masson, Adam Shlien, Stefan T. Palsson, Michael L. Frigge, Thorgeir E. Thorgeirsson, Jeffrey R. Gulcher, and Kari Stefansson. A high-resolution recombination map of the human genome. *Nature Genetics*, 31:241–247, July 2002.

[14] Leonid Kruglyak, Mark J. Daly, Mary Pat Reeve-Daly, and Eric S. Lander. Parametric and nonparametric linkage analysis: A unified multipoint approach. *The American Journal of Human Genetics*, 58:1347–1363, 1996.

[15] Leonid Kruglyak and Eric S. Lander. Faster multipoint linkage analysis using fourier transforms. *Journal of Computational Biology*, 5:1–7, March 1998.

[16] Eric S. Lander and Philip Green. Construction of multilocus genetic linkage maps in humans. *Proceedings of the National Academy of Sciences of the United States of America*, 84:2363–2367, April 1987.

[17] Jing Li and Tao Jiang. Efficient rule-based haplotyping algorithm for pedigree data. In *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 197–206, April 2003.

[18] Jing Li and Tao Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. In *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 101–110, March 2004.

[19] Shin Lin, Aravinda Chakravarti, and David J. Cutler. Haplotype and missing data inference in nuclear families. *Genome Research*, 14:1624–1632, August 2004.

[20] Shin Lin, David J. Cutler, Michael E. Zwick, and Aravinda Chakravarti. Haplotype inference in random population samples. *The American Journal of Human Genetics*, 71:1129–1137, November 2002.

[21] Jonathan Marchini, David Cutler, Nick Patterson, Matthew Stephens, Eleazar Eskin, Eran Halperin, Shin Lin, Zhaohui S. Qin, Heather M. Munro, Gonçalo R.

Abecasis, and Peter Donnelly. A comparison of phasing algorithms for trios and unrelated individuals. *The American Journal of Human Genetics*, 78:437–450, March 2006.

[22] Kyriacos Markianos, Mark J. Daly, and Leonid Kruglyak. Efficient multipoint linkage analysis through reduction of inheritance space. *The American Journal of Human Genetics*, 68:963–977, 2001.

[23] Tianhua Niu. Algorithms for inferring haplotypes. *Genetic Epidemiology*, 27:334–347, December 2004.

[24] Tianhua Niu, Zhaohui S. Qin, Xiping Xu, and Jun S. Liu. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *The American Journal of Human Genetics*, 70:157–169, January 2002.

[25] Jeffrey R. O'Connell and Daniel E. Weeks. Pedcheck: a program for identification of genotype incompatibilities in linkage analysis. *The American Journal of Human Genetics*, 63:259–266, 1998.

[26] Matthew Stephens and Peter Donnelly. A comparison of bayesian methods for haplotype reconstruction from population genotype data. *The American Journal of Human Genetics*, 73:1162–1169, 2003.

[27] The International HapMap Consortium. A second generation human haplotype map of over 3.1 million snps. *Nature*, 449:851–861, October 2007.

[28] Janis E. Wigginton and Gonçalo R. Abecasis. Pedstats: descriptive statistics, graphics and quality assessment for gene mapping data. *Bioinformatics*, 21:3445–3447, 2005.