dna.ancestry.com
http://dna.ancestry.com/#/matches/74039570-7118-4191-9FBA-E0F507DD17EF?filterBy=ALL&sortBy=RELATIONSHIP&page=1

DNA Tests for Ethnicity & Genealogical DNA testing at AncestryDNA

Matching White Paper: Discovering IBD Matches across a large, growing database

Catherine A. Ball, Mathew J. Barber, Jake K. Byrnes, Peter Carbonetto, Kenneth G. Chahine, Ross E. Curtis, Julie M. Granka, Eunjung Han, Amir Kermany, Natalie M. Myres, Keith Noto, Jianlong Qi, and Yong Wang (in alphabetical order)

1. Introduction

AncestryDNA offers several genetic analyses to help customers find, preserve, and share their family history.

The AncestryDNA analysis that is the topic of this document is known in the population genetic literature as identity-by-descent (IBD) analysis, or IBD matching. IBD analysis identifies pairs of customers who have long identical genetic segments suggestive of a recent common ancestry. Given the amount of identical DNA between two members, the analysis estimates the possible degree of relationship between the pair. By drawing connections between putative relatives, IBD analysis offers the opportunity for AncestryDNA members to expand their documented pedigrees. Beyond its immediate interest, IBD matching is central to AncestryDNA's analysis to identify Circles, a collection of descendants of a given ancestor (see AncestryDNA's Circles White Paper).

In this paper, we describe in detail the processes we use to identify and interpret segments of DNA shared identical-by-descent between individuals. We begin with an introduction to our data and the necessary subtasks, and then describe each of these subtasks in detail in the sections that follow.

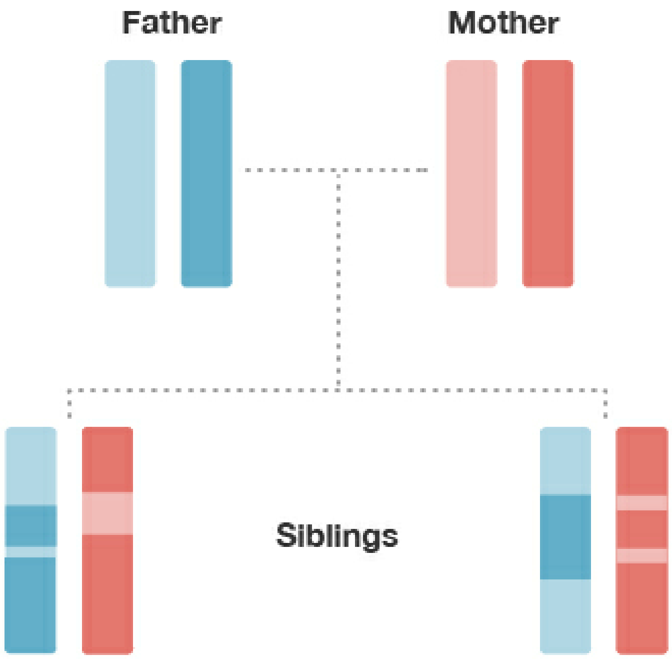
1.1 What is IBD?

In the family in Figure 1.1, we illustrate each person in the pedigree with a representation of his or her genome -- a colored bar running from top to bottom representing a sequence of DNA. Humans are a diploid species -- that is to say, each person has two copies of the human genome, one inherited from his or her father and one inherited from his or her mother. Each copy of the inherited genome is called a haplotype. In Figure 1.1, the father inherited the light blue haplotype from one of his parents, and the dark blue haplotype from the other. The mother inherited the light and dark red haplotypes from each of her parents. As Figure 1.1 indicates, a child is a random mixture of haplotypes from each parent. Note that each parental haplotype is a mosaic of haplotypes that the parent has inherited from her/his own parents. This is due to a process called recombination. Thus, each individual in Figure 1.1 is represented by two colored bars, each being the haplotype they inherited from their parents.

Figure 1.1: One copy of a person's genome is represented as a colored bar, or haplotype, where the same color indicates the same sequence of DNA. This pedigree gives example haplotypes of the two children of two parents. Each child has a randomized mixture of sequences from his or her parents.

If we line up the siblings' genomes from top to bottom (in Figure 1.1), we see that they both have two haplotypes, one light or dark blue (from the father), and one light or dark red (from the mother). Notice that the two siblings often have the same color of DNA in the same place on the genome. That means they have the same exact sequence of DNA in those places. These segments of DNA are said to be identical by descent (IBD), i.e. identical because they were inherited from a common ancestor (in this case two common ancestors, the parents).

The more distant the common ancestors two people share, the shorter and fewer IBD segments they will have. For example, Figure 1.2 shows three distant descendants of a common pair of ancestors. These descendants have each inherited only a small proportion of their DNA from the common ancestors (in Figure 1.2, the "grey" DNA was inherited from other ancestors that they do not have in common).



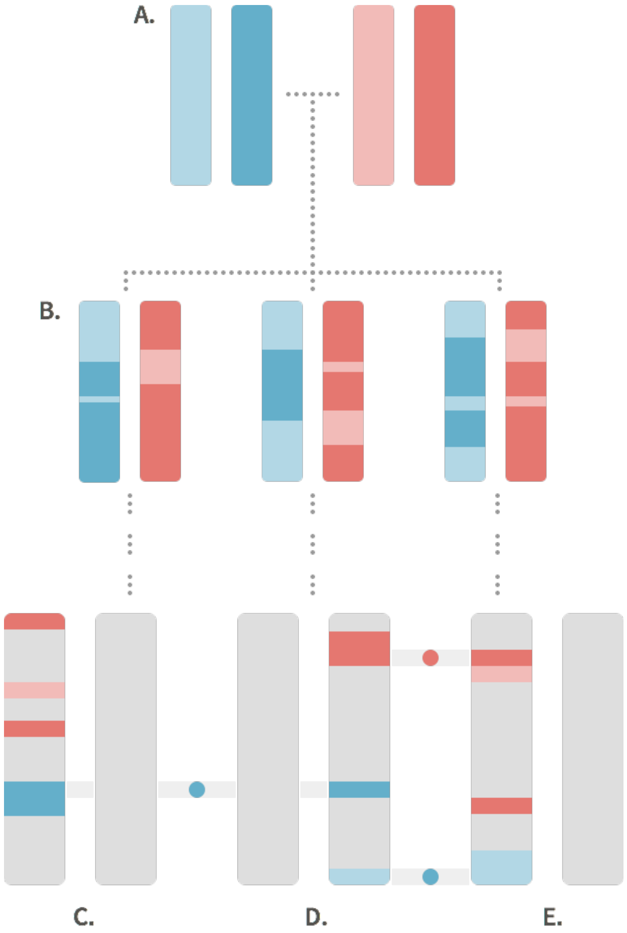


Figure 1.2: DNA that is identical-by-descent among distant cousins. See legend of Figure 1.1.

While the fifth cousins in Figure 1.2 have all inherited some DNA from their common ancestors, only a few haplotypes are identical in the same place on the genome. In one case (the rightmost and leftmost genomes in Figure 1.2), two fifth cousins do not share any IBD segments at all.

The main goal of DNA matching is to identify segments of the genome that are identical by descent between pairs of individuals. Given these inferred IBD segments, the goal is then to estimate the number of generations separating any two individuals (described in detail in Section 5).

However, as there are major challenges in identifying IBD segments, IBD estimation requires a sophisticated approach. Here, we introduce those challenges and the methods developed to address them.

1.2 Phasing

The first challenge is that the genotype data we collect are not separated into distinct sequences representing parental haplotypes, but rather consist of a pair of nucleotides for each position on the sequence. To understand what this means, consider the following example genotype data in Table 1.3. These data represent genotype calls for a series of locations across the genome. Each of these locations is chosen specifically because they are known to have common variation in humans (for more details, see AncestryDNA's Ethnicity Estimate White Paper). These variations are called single nucleotide polymorphisms (SNPs). Since all people have two copies of the genome (one from the mother and one from the father), there are two alleles at each SNP location.

Genetic Marker	Allele Type #1 (Less Frequent)	Allele Type #2	Copies of Allele	Copies of Allele Type #2
1	A	A	0	2
2	C	C	0	2
3	A	A	1	1
4	C	C	1	1
5	G	G	0	2
6	T	T	0	2
7	A	A	2	0
8	C	C	1	1

Table 1.3: A small amount of genotype data for a single person.

Table 1.3 shows eight sequential SNPs. The first SNP is in a location where some humans have an A, and others have a G. The genotype shown in the table has two copies of G--in other words, the individual inherited a G from both parents, and is thus homozygous (i.e., has identical alleles at a given SNP). The individual is also homozygous at the second position, where the person has two A alleles. At the third genetic marker, this individual is heterozygous, meaning the individual has a different nucleotide on each of their chromosomes: one A and one G. This person is also heterozygous (one C and one A) at the fourth genetic marker.

This creates a challenge, because we do not know whether the parent (and thus the individual's haplotype) with the C at genetic marker #4 has the A or the G at marker #3. In other words, for this pair of heterozygous sites, we do not know whether the C and the A, or the C and the G, were inherited together from the same parent.

Not knowing which alleles an individual inherited together from a parent leads to issues when we try to identify IBD matches. This is because an IBD match will only be on one genomic haplotype inherited from either the mother or the father (see Figure 1.2). However, since our data do not directly indicate each haplotype sequence, it must be computationally inferred. The process of determining the two distinct haplotype sequences of an individual is called phasing in the literature. In Section 2, we present and describe our phasing approach, called Underdog.

1.3 Finding Segments of Identical DNA Between Pairs

The second major challenge is finding identical DNA sequences between any pair of samples in a large and continuously-growing database. At the time of writing this document, the AncestryDNA database was made up of over 500,000 genotyped samples, and thus about 125 billion pairs of samples to check for IBD. To do these comparisons efficiently, we use a proprietary implementation of GERMLINE [Gusev et al ,GenRes 2009], which uses a hashing scheme to avoid explicitly making hundreds of billions of pairwise comparisons.

While the existing GERMLINE implementation was intended to be run once on a finalized (and relatively small) data set, we require an implementation that is able to facilitate a continuous incoming stream of new DNA data samples. To that end, we have designed and implemented a database and software called J-Germline. J-Germline stores phased genotype data of existing samples and processes small batches one at a time. We describe J-Germline in detail in Section 3.

1.4 Filtering Out Non-IBD Segments

A third major challenge is to filter out instances where two people may have an identical segment of DNA for a reason other than because they have a recent common ancestor. Firstly, it is possible that two people share identical DNA because of ancient (and not recent) shared history (or genealogy) and, secondly, it is possible that the observation of identical DNA between two people is false.

The longer the stretches of evidence for identical haplotypes the more evidence there is that the identity is due to a recent common ancestor. However, the quantification of the evidence varies across the genome and has to adjust to our observation of genetic variation (where some regions might have sparse genetic markers, low phasing accuracy, genotype errors, etc.) and the properties of the underlying genetic variation itself (where some regions might have undergone selective pressure, and/or some regions might have varying genetic diversity within different ethnic groups, etc.).

Our goal is to identify only the identical segments that are IBD due to a recent common ancestor. AncestryDNA's large customer database has enabled us to improve our statistical models to remove false positive IBD segments from the results from J-Germline. To this aim, we have developed a new statistical method called Timber, which we describe in Section 4.

1.5 Estimating Relationships

Finally, a fourth major challenge is interpreting the results -- that is, translating the length and number of IBD segments shared between individuals into their estimated relationship. Identical twins and parent-child pairs will share DNA that is IBD across the entire genome, but a pair of distant cousins will share significantly less DNA. However, the amount of DNA that distant cousins share IBD varies greatly (and randomly) from pair to pair. This variability means that there are cases where we are less certain of the exact relationship (e.g., the number of generations back one would have to go to find a common ancestor) based on IBD alone.

We have examined thousands of pairs of individuals with known relationships and estimated IBD between them to inform our estimation of the degree of separation between a pair of samples. We have also performed simulations based on known or estimated recombination rates (e.g., HapMap Nature 2007]) in the human genome to measure the variance in IBD between pairs of related individuals. From this, we have developed a method (described in detail in Section 5) to make accurate estimates of the nature of the relationship between each customer and their IBD matches.

Figure 1.4 shows the basic steps of our IBD analysis pipeline, and indicates the flow of data between the algorithms described above

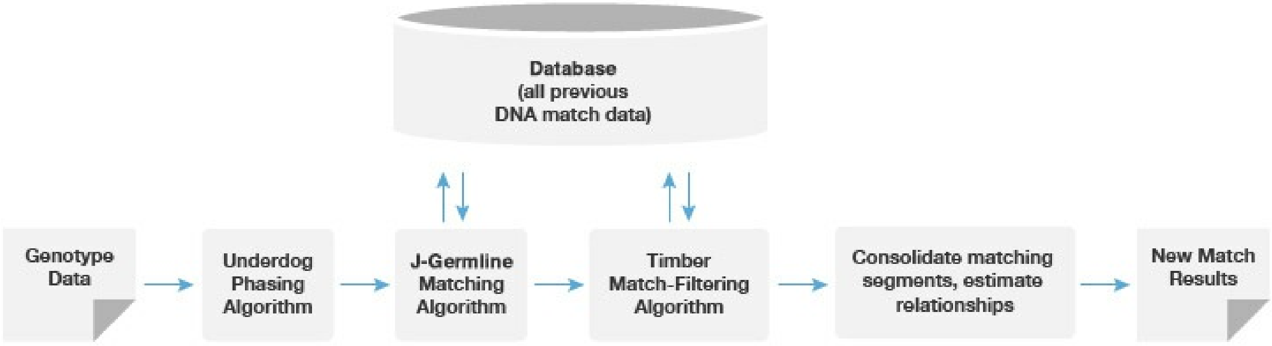

Database
(all previous
DNA match data)
Genotype Data
Underdog Phasing Algorithm
J-Germline Matching Algorithm
Timber Match-Filtering Algorithm
Consolidate matching segments, estimate relationships
New Match Results

Figure 1.4: Overview of the Matching 2.0 pipeline. Underdog is our phasing program, J-Germline is our matching program, and Timber is our match analysis program.

2 Underdog Phasing Algorithm

2.1 Introduction

As we describe in Section 1.2, genotype data alone does not reveal the phase of a given sample's genotype -- that is, it does not distinguish the two haplotypes that a person inherited from each of their parents. Phasing an individual's genotype -- i.e., separating diploid genotypes into a pair of haplotype chromosomes (see Figure 1.3) -- can be performed easily and with high accuracy given genotype data from two parents and a child (known as "trio" phasing), or from one parent and a child ("duo" phasing). In these cases, the alleles transmitted to a child from each of their parents can be easily inferred. However, for sites where phase is ambiguous even given parental data, or when parent or child genotype data is not available, phasing must be performed in a different way entirely by building what are called haplotype models.

These methods that phase genotypes without requiring parental information build statistical models of the haplotypes that are likely to be observed in a population. Such models are described in detail in the following sections. Generally, a statistical haplotype model is built in part using a set of training samples -- a set of haplotypes from individuals who are already phased. In some cases, the models are also iteratively built using the unphased genotypes of samples whose phase is desired. Using statistical models in the methods described below, the combination of previously phased and unphased genotypes can be used to computationally produce an estimate of the phase for the desired unphased genotype.

Such algorithms that phase in this manner traditionally do so by phasing many samples together, exploring the possible haplotype assignments in the statistical model, and iteratively improving the phase [Browning NatGen 2011, Browning AJHG 2007, Williams AJHG 2012]. There are many different algorithms for phasing that have been explored [Browning NatGen 2011], but all have the property that a larger input set will increase phasing accuracy. However, when the input contains hundreds of thousands of samples, these algorithms become intractable, forcing users to discard potentially useful data. Furthermore, the entire process generally must be repeated to phase new samples.

2.2 The Beagle Phasing Algorithm

In developing the Underdog phasing algorithm, we began by creating statistical models similar to those developed by [Browning AJHG 2007] in a method called Beagle. As described above, Beagle learns a statistical model to summarize the haplotypes likely to be observed in a population. Building this statistical model allows one to obtain an estimate of haplotype phase for a desired set of customer genotypes. (Note that since each chromosome is inherited independently, such a model of haplotypes is generally done at the level of a chromosome at a time.)

To build this statistical model of chromosomal haplotypes, Beagle [Browning AJHG 2007] breaks each chromosome into a series of "windows" (e.g., 500 SNPs each) and builds a haplotype Markov model (see Figure 2.1) that represents a probability distribution over haplotypes in each window. The nodes in the model represent clusters of common haplotypes, and the edges emit either the allele type #1 or type #2 and transition to a node at the next level (i.e., SNP) in the model.

From each haplotype model, Beagle creates a diploid hidden Markov model (HMM) where each state in the HMM represents an ordered pair of states in the haplotype cluster Markov model--thus a path through the diploid HMM represents two paths through the haplotype model. Therefore, the paths through the diploid HMM that are consistent with a given genotype (i.e., have the same allele variants at each level of the model) represent a probability distribution over possible ways of phasing that genotype. For further information about Markov and hidden Markov models, see [Rabiner IEEE 1989].

The Beagle algorithm (refer to Algorithms 2.1, 2.2 and 2.3) takes as input (i) a reference set R of already-phased genotypes, and (ii) a query set U of unphased genotypes. It starts by assigning an arbitrary (e.g., random) phase to U (typically including multiple ways of phasing each genotype) and then learns a new set of models (in Figure 2.1) from the resulting haplotypes and from R. The models are used to probabilistically generate new phased versions of U (again, Beagle typically samples multiple sets of haplotypes per genotype). The process repeats over a number of iterations until the models are finalized and then used to compute the most likely phase for each genotype in U (in the HMM literature, this is known as the Viterbi path through the diploid HMM). The final phased result is a combination of the phased results from each window (for details, see [Browning AJHG 2007]).

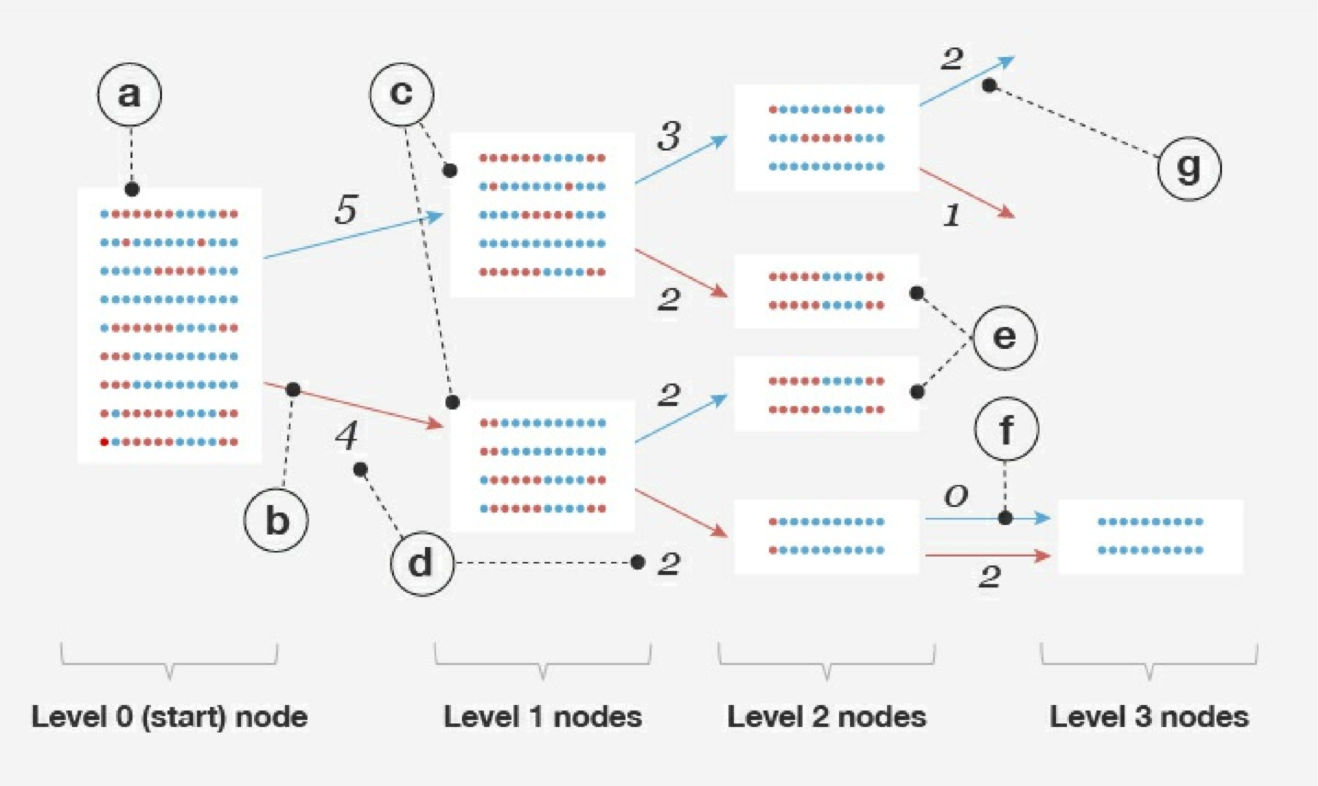


Figure 2.1: A representation of a haplotype cluster Markov model during Beagle or Underdog' model-learning process. (a) Each node contains a cluster of haplotypes, represented here by an sequence of colored dots (e.g., let blue represent the allele type #2 and red represent the allele type #1). The start state contains all the haplotypes in the training set. (b) Each node has up to two outgoing transitions, for the type #1 and type #2 alleles. A transition to a node at level d will split up a haplotype cluster based on the SNP at position d in the haplotype. (c) The haplotype clusters at level 1 that result from splitting on the first allele. Note that only the alleles after the first are shown in the clusters, as the merging process at level d is concerned only with the distributions of the haplotypes that follow the dth SNP. (d) We keep track of the allele counts for each transition. They will determine the transition probabilities. (e) These two level d=2 nodes will be merged during the learning process, because the distribution of haplotypes is similar enough (in the figure, they are identical). (f) Beagle models do not have edges for haplotypes that do not appear in the training set (e.g., red, red, blue) but in Underdog we will consider an edge with a haplotype count of zero to be connected to the same node as the alternative allele. When we initialize the model's transition probabilities, we assign a nonzero likelihood to such haplotypes (essentially explaining away such haplotypes as having at least one genotyping error). (g) We continue to split and merge nodes (see Algorithm 2) until all D alleles in the haplotypes are represented by a level in the model. Level D will always be a single terminal node.

The procedure Beagle uses to build models from a set of example haplotypes (R) comes from Ron et al. [Ron JCompSysSci 1998] and is described in [Browning AJHG 2006]. Each model is built by splitting nodes at level d into two children each, one for each possible allele at that level, to create the nodes at level d+1. Then, nodes at level d+1 whose haplotype clusters have similar enough distributions of haplotypes are merged together. After merging all such pairs of nodes, the model-building procedure moves on to the next level. Figure 2.1 illustrates a model during the building process, and Algorithms 2 and 3 outline the procedure.

The model-building process is time-consuming, increasingly so with the sizes of R and U (see Tables 1 and 2), and the models of Beagle cannot be reused to phase new samples. This means that new unphased samples cannot directly benefit from previously-phased samples or from the models that phased them. We present an alternative approach, wherein we learn models once from a large collection of haplotypes, save the learned models to disk, and then use the model to quickly phase any number of unphased samples thereafter. We describe our approach Underdog in detail, and contrast it with Beagle in the next section.

Algorithm 1 *An overview of the Beagle algorithm. For details, see [1]. Input is a reference set* R *of phased samples, and A set of unphased samples,* U*.* R *is the number of times per genotype in* U *to sample from a model. A typical value would be* $R = 4$*. diploid-HMM-sample and diploid-HMM-Viterbi refer to either sampling or computing the most likely path from a diploid HMM, as described in the text.*

```
procedure Beagle $(R, U)$
    Phase each sample in $U$ arbitrarily (e.g., randomly) $R$ times each and call the result $P$
    for iteration $i$ in [1,2,...max-iterations] do
        $M \leftarrow$ model$(R \bigcup P)$ // Learn haplotype cluster Markov model (see Algorithms 2 and 3
        $P \leftarrow$ ⊠
        for $u$ ⊠ $U$ do
            if $i$ =max-iterations then
                // For the last iteration, set the final phase of $u$ from the Viterbi path in the diploid-HMM
                $P \leftarrow P \bigcup$ diploid-HMM-Viterbi $(M, u)$
            else
                // Sample $R$ possible ways of phasing $u$ from $M$
                for $i$ in [1,2,...R] do
                    $P \leftarrow P \bigcup$ diploid-HMM-sample $(M, u)$
    return $P$
```

Algorithm 2 *The model learning algorithm. Haplotypes are represented as bit sequences (i.e., alleles are 0 or 1). The function compare decides if two haplotype clusters are similar enough to merge together. The version of compare that Beagle uses is described in Algorithm (TODO). The version of compare that we use is described in Algorithm 4. split (Algorithm 5) is a simple subroutine that divides a haplotype set into two subsets based on the value of one of its alleles. Input to model is* N *training haplotypes* X*, each consisting of* D *consecutive SNPs. Output is A haplotype Markov model consisting of* $d + 1$ *levels, each a set of nodes that transition to nodes in the next level.*

```
procedure model $(X)$
    levels$_0 \leftarrow$ node(haplotypes $= X$, parents $=$ ⊠)
    for each level $d$ in $[1, 2, \ldots, D]$ do
        levels$_d \leftarrow$ ⊠
        for each node $x$ in levels$_{d-1}$ do
            $S \leftarrow$ split $(x$.haplotypes $, d)$ // (split up node's haplotypes by allele $d$, see Algorithm 5)
            $x$.children$_0 \leftarrow$ node(haplotypes $= S_0$, parents $= \{x\}$
            $x$.children$_1 \leftarrow$ node(haplotypes $= S_1$, parents $= \{x\}$
        $Q \leftarrow$ ⊠// initialize [priority] queue to empty
        for each pair of nodes $x, y$ in levels$_d$ do
            similar, score $\leftarrow$ compare $(x$.haplotypes$, y$.haplotypes$, |x$.haplotypes$|, |y$.haplotypes$|, d, D)$
            if similar then
                enqueue $(Q, x, y$, score $)$// enqueue pair $x, y$
        while $Q$ is not empty do
            $x, y$, score $\leftarrow$ pop$(Q)$// get most similar nodes $x, y$
            if $x$ and $y$ are still in levels$_d$ then
                levels$_d \leftarrow$ levels$_d \setminus \{x, y\}$// remove $x$ and $y$ from level $d$
                // and merge nodes $x$ and $y$:
                $z \leftarrow$ node(haplotypes $= x$.haplotypes $\bigcup y$.haplotypes, parents $= x$.parents $\bigcup y$.parents $)$
                for each node $p$ in $z$.parents do
                    replace $x$ and $y$ with $z$ in $p$.children
                for each node $w$ in levels$_d$ do
                    similar, score $\leftarrow$ compare $(z$.haplotypes$, w$.haplotypes$, |z$.haplotypes$|, |w$.haplotypes$|, d, D)$
                    if similar then
                        enqueue $(Qz, w$, score $)$
                levels$_d \leftarrow$ levels$_d \bigcup \{z\}$// finally, add $z$ to level $d$
        return levels // return model
```

Algorithm 3 *The algorithm Beagle uses for comparing two model nodes (In our approach, we replace this with Algorithm 4). Input is set of haplotypes X (bit sequences all of length D) of size n_x, and a set Y of size n_y that represent the haplotype clusters in two nodes at level d in a model that has D levels. m and b are pre-defined constants. Output is (i) Whether the two nodes are similar enough to merge, and (ii) if so, a similarity score.*

```
procedure compare $(X, n_x, Y, n_y, d, D)$
    if $d > D$ then
        return (similar = true, score = 0) // no more alleles to compare
    $Sx \leftarrow$ split $(X, d)$ // split X according to the allele at SNP d
    $Sy \leftarrow$ split $(Y, d)$ // also Y
    $a \leftarrow m \times \sqrt{\frac{1}{n_x} + \frac{1}{n_y}} + b$ // Threshold based on size of haplotype clusters
    $diff_0 \leftarrow \frac{|Sx_0|}{n_x} - \frac{|Sy_0|}{n_y}$ // difference in proportion between these haplotypes
    if $diff_0 \geq a$ then
        return (similar = false, score = n/a)
    $diff_1 \leftarrow \frac{|Sx_1|}{n_x} - \frac{|Sy_1|}{n_y}$
    if $diff_1 \geq a$ then
        return (similar = false, score = n/a)
    if $max(\frac{|Sx_0|}{n_x}, \frac{|Sy_0|}{n_y}) \geq a$ then
        $diff_0 \leftarrow max(diff_0, compare(Sx_0, n_x, Sy_0, n_y, d+1, D))$
        if $diff_0 \geq a$ then
            return (similar = false, score = n/a)
    if $max(\frac{|Sx_1|}{n_x}, \frac{|Sy_1|}{n_y}) \geq a$ then
        $diff_1 \leftarrow max(diff_1, compare(Sx_1, n_x, Sy_1, n_y, d+1, D))$
        if $diff_1 \geq a$ then
            return (similar = false, score = n/a)
    return (similar = true, score = max$(diff_0, diff_1)$)
```

2.3 The Underdog phasing algorithm

In contrast to Beagle, our goal is to learn models from large training sets and use them to phase samples efficiently, after the models are finalized. This difference presents a few new challenges that Beagle does not have to consider. The adjustments we make to address these challenges distinguish Underdog from the methods of Beagle.

First, Beagle's models only represent haplotypes that appear in the training examples. We need to be able to phase new samples (not in the training data), potentially with haplotypes or genotypes that are not part of the training set when the models are built, and to do so without rebuilding the models. To handle this, instead of setting transition probabilities directly from the allele counts (Figure 2.1(d)) associated with a node's outgoing edges as Beagle does, we set the transition probability for allele variant a to

(Eqn. 2.1)

$$P(a) = \frac{1}{2}\gamma + \frac{n_a}{n_a + n_{\bar{a}}}(1 - \gamma)$$

where n_a is the count of allele a and $n_{\bar{a}}$ is the count of the alternative allele. Here, γ is the user-specified probability (e.g., of a genotype error, in which case either allele is equally likely). For example, in Figure 2.1, consider the bottom state in level 2. Instead of having only one transition (to the bottom state in level 3) with 100% probability of the allele type #1 (red, in the figure), we would add a second transition for the allele type #2 (blue, also to the bottom state in level 3), but instead of assigning it zero probability, we would assign it γ probability . We adjust all transition probabilities this way, but they differ significantly from the probabilities Beagle would assign only when one of the alleles is particularly rare. By adjusting the transition probabilities according to (Eqn. 2.1), our haplotype Markov models represent a likelihood for all possible haplotypes, and thus the corresponding diploid HMMs can phase any genotype, even one that contains haplotypes that do not appear in the model training set.

A second challenge is that building models from hundreds of thousands of haplotypes creates very large models (e.g., millions of states), which has a large effect on phasing time because the diploid HMMs are quadratically larger than the haplotype models themselves. To handle this, we take advantage of the fact that most of the possible ways of phasing a sample are extremely unlikely given the haplotype cluster models and most of the likelihood mass is contained in a relatively small subset of possible paths through the model. Therefore, when computing the Viterbi path through a diploid HMM, we discard the most unlikely pairs of haplotype states at each level until we have discarded ε of the likelihood mass at that level. The procedure only differs from that described in [Browning AJHG 2007], in that the likelihood associated with many of the states in the diploid HMM is set to zero and the state is effectively removed from the model. Figure 2.2 shows the phasing time required for various values of ε.

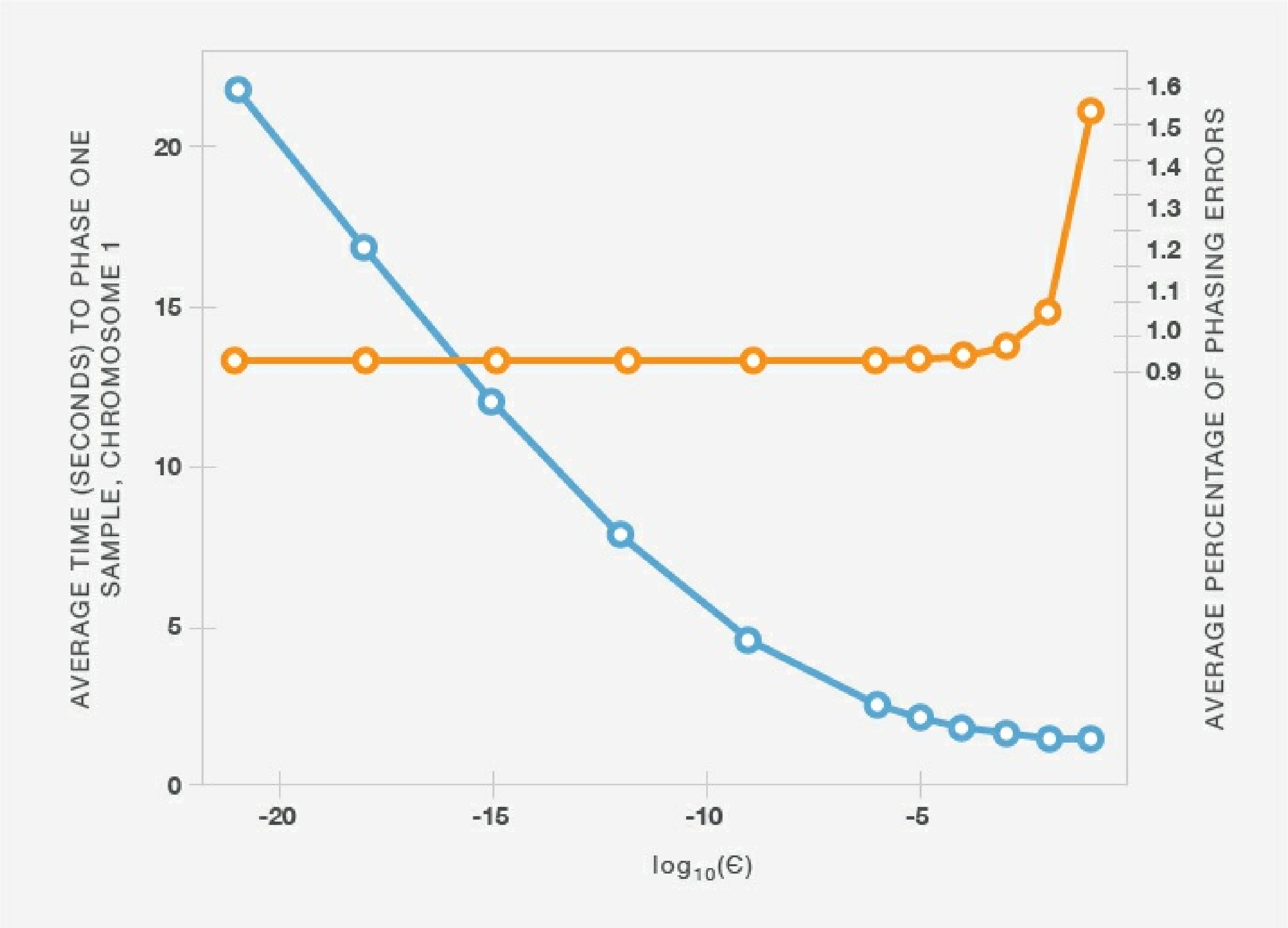


Figure 2.2: Relationship between the diploid HMM Viterbi calculation parameter ε and the number of processor seconds to phase chromosome 1. If we set ε=0, the average sample phasing time is 63.17 seconds, and the average phasing error rate is 0.93% (about the same as for all ε ≤ 10e^-9). Note that the time reported is phasing time only, and does not include file I/O or recombining phased segments together.

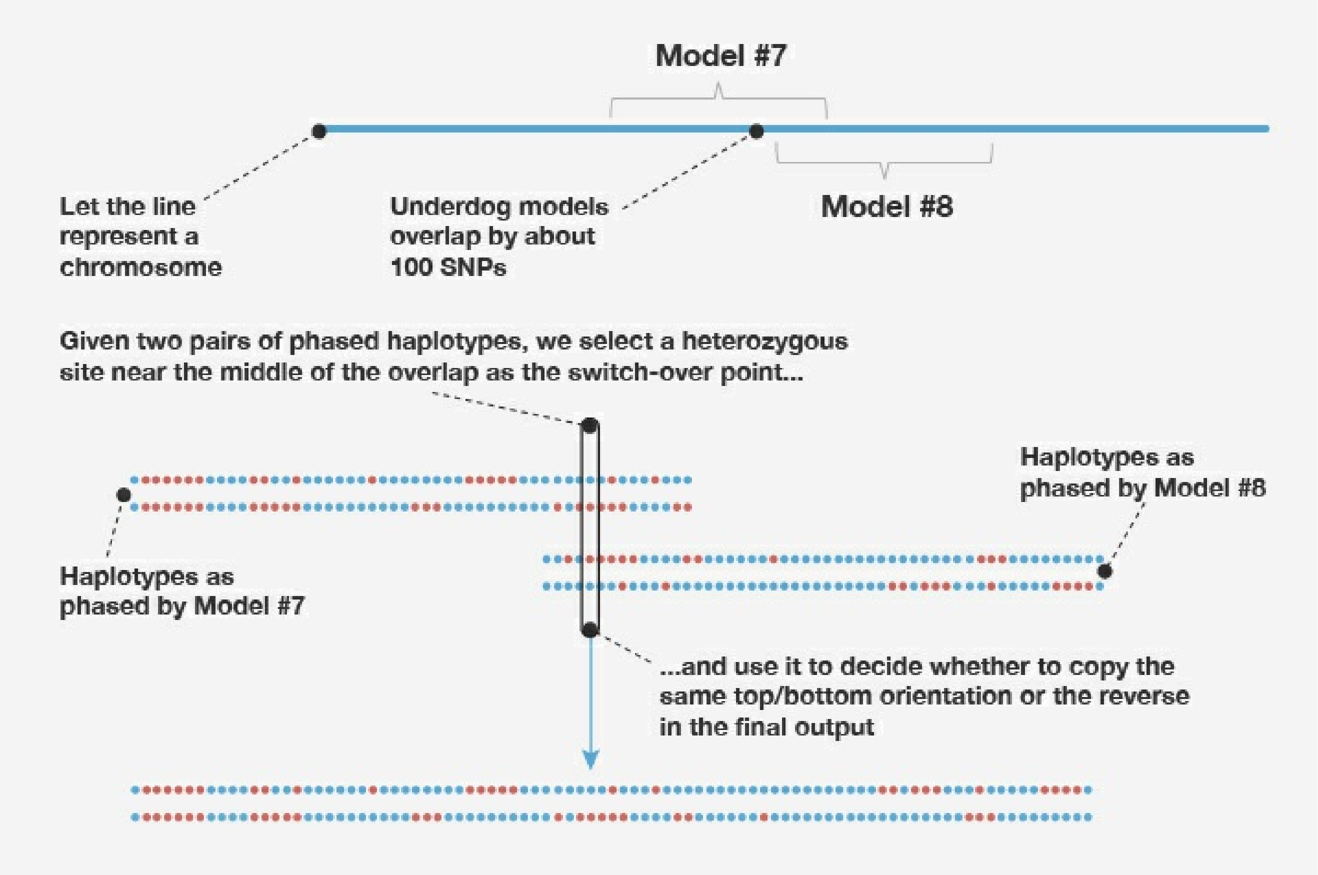

Model #7
Let the line represent a chromosome
Underdog models overlap by about 100 SNPs
Model #8
Given two pairs of phased haplotypes, we select a heterozygous site near the middle of the overlap as the switch-over point...
Haplotypes as phased by Model #8
Haplotypes as phased by Model #7
...and use it to decide whether to copy the same top/bottom orientation or the reverse in the final output

Figure 2.3: In order to ensure that every phasing decision is made in part of a model that has sufficient SNPs both upstream and down, we create models that overlap in a portion of the genome they cover. We copy the phase from each model to the output and crossover to the next model at the point of the heterozygous site closest to the midpoint of the overlapping region.

Because we divide each chromosome into relatively short windows and phase once, a third challenge is that we must deal with the fact that the phase among the first and last several SNPs in each model's window may be less accurate because they have shorter upstream or downstream haplotypes, respectively, from which to learn. To handle this, we create models that overlap by a significant amount (e.g., 100 SNPs). When we copy each model's output to the final Viterbi phased haplotype, we switch between one model and the next at the heterozygous site closest to the middle of the overlapping region (see Figure 3) so that phasing decisions made near one of the extreme ends of any model is never part of the phase.

A fourth challenge is deciding when to merge two model nodes (haplotype clusters at the same level) during the model-building procedure. The motivation for doing so is to reduce the size and bias of the models by consolidating clusters of haplotypes that come from the same distribution (e.g., Figure 2.1e). Therefore, we want to keep two nodes X and Y as separate nodes in the model if there exists a haplotype h of any length that is more likely to belong to one than the other; i.e., P(h|X)≠P(h|Y). Let n_{x} and n_{y} be the respective number of haplotypes in X and Y, and let ??? and ???? be the observed occurrences of h in X and Y, respectively. The frequency of h in X and Y is estimated to be $p^{\{h\}}_{x}$ and $p^{\{h\}}_{y}$, respectively. The approach of Ron et al. [Ron JCompSysSci 1998] will not merge nodes if the following holds true for any h and a pre-defined constant, α:

(Eqn. 2.2)

$$\left|\hat{p}_x^{(h)} - \hat{p}_y^{(h)}\right| \geq \alpha$$

If two clusters do indeed represent the same haplotype distribution, the algorithm is more likely to observe a large difference in haplotype frequencies if the sample size is small. Beagle [Browning AJHG 2006] accounts for this by replacing the constant α with a term that depends on sample sizes:

(Eqn 2.3)

$$\left|\hat{p}_x^{(h)} - \hat{p}_y^{(h)}\right| \geq \sqrt{\frac{1}{n_x} + \frac{1}{n_y}}$$

In Underdog, we instead reformulate the inequality as a formal hypothesis test that accounts for the haplotype proportions themselves as well as the difference between them. Our formulation is based on Welch's t-test [Welch Biometrika 1947] which is, in the form of the inequalities above,

$$\left|\hat{p}_x^{(h)} - \hat{p}_y^{(h)}\right| \geq C^2 \sqrt{\frac{\hat{p}_x^{(h)}(1 - \hat{p}_x^{(h)})}{n_x} + \frac{\hat{p}_y^{(h)}(1 - \hat{p}_y^{(h)})}{n_y}}$$

(Eqn 2.4)

where C is a pre-defined constant. The concern with Welch's t-test is that it is too confident in its estimation of variance when the frequency estimate is close to 0 or 1. To avoid this problem, we regularize the frequency estimate using a symmetric beta distribution as a prior. Thus, we replace the $\hat{p}$ estimates with their posteriors:

(Eqn 2.5)

$$\tilde{p}_x^{(h)} = \frac{n_x(h) + \alpha}{n_x + \alpha + \beta}$$

(Eqn 2.6)

$$\tilde{p}_y^{(h)} = \frac{n_y(h) + \alpha}{n_y + \alpha + \beta}$$

and rewrite (Eqn 2.4) as

$$\left|\tilde{p}_x^{(h)} - \tilde{p}_y^{(h)}\right| \geq C^2 \sqrt{\frac{\tilde{p}_x^{(h)}(1 - \tilde{p}_x^{(h)})}{n_x} + \frac{\tilde{p}_y^{(h)}(1 - \tilde{p}_y^{(h)})}{n_y}}$$

(Eqn 2.7)

or equivalently

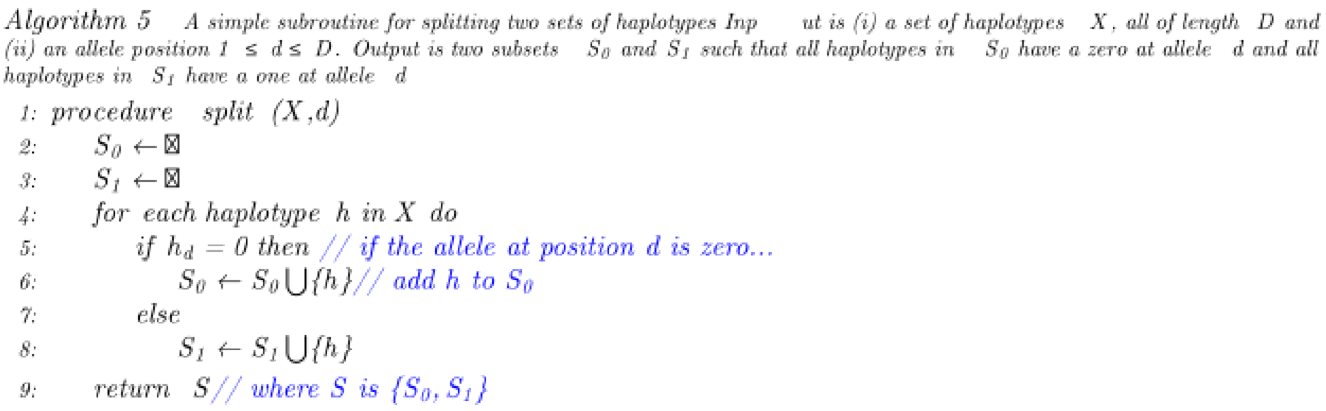
$$\frac{(\tilde{p}_x^{(h)} - \tilde{p}_y^{(h)})^2}{\frac{\tilde{p}_x^{(h)}(1-\tilde{p}_x^{(h)})}{n_x} + \frac{\tilde{p}_y^{(h)}(1-\tilde{p}_y^{(h)})}{n_y}} \geq C$$

(Eqn 2.8)

We find $\alpha=\beta=\frac{1}{2}$ to be reasonable parameters for the beta distribution. C is a function of the p-value we wish to apply to the hypothesis that X and Y represent the same haplotype distribution. We choose C using validation sets to optimize model size (which influences phasing speed) and accuracy. Algorithm 2.4 (together with Algorithm 2.2) describes in detail how we apply (Eqn 2.8) to all the haplotypes in X and Y during the model building procedure.

Algorithm 4 *The algorithm we use (replacing Algorithm 3) while learning a haplotype cluster Markov model to compare two model nodes and decide if they should be merged together. Input is set of haplotypes X (bit sequences all of length D) of size n_x, and a set Y of size n_y that represent the haplotype clusters in two nodes at level d in a model that has D levels. Output is (i) Whether the two nodes are similar enough to merge, and (ii) if so, a similarity score. α and β of (the beta distribution) and C (a threshold) are pre-defined parameters.*

```
procedure compare $(X, n_x, Y, n_y, d, D)$
    if $d > D$ then
        return (similar = true, score = 0) // no more alleles to compare
    $Sx \leftarrow$ split $(X, d)$ // split $X$ according to the allele at SNP $d$
    $Sy \leftarrow$ split $(Y, d)$ // also $Y$
    // First, see if $Sx_0$ and $Sy_0$ are different
    $px_0 \leftarrow (|Sx_0| + \alpha)/(n_x + \alpha + \beta)$
    $py_0 \leftarrow (|Sy_0| + \alpha)/(n_y + \alpha + \beta)$
    $score_0 \leftarrow ((px_0 - py_0) \times (px_0 - py_0))/((px_0 \times (1 - px_0))/n_x + (py_0 \times (1 - py_0))/n_y)$
    if $score_0 \geq C$ then
        return (similar = false, score = n/a)
    // Now try comparing haplotypes with a "1" at allele $d$
    $px_1 \leftarrow (|Sx_1| + \alpha)/(n_x + \alpha + \beta)$
    $py_1 \leftarrow (|Sy_1| + \alpha)/(n_y + \alpha + \beta)$
    $score_1 \leftarrow ((px_1 - py_1) \times (px_1 - py_1))/((px_1 \times (1 - px_1))/n_x + (py_1 \times (1 - py_1))/n_y)$
    if $score_1 \geq C$ then
        return (similar = false, score = n/a)
    // Compute the highest score we could get in the recursion that follows
    $lowpx_0 \leftarrow \alpha/(n_x + \alpha + \beta)$
    $lowpy_0 \leftarrow \alpha/(n_x + \alpha + \beta)$
    $maxscore_0 \leftarrow max\left(\frac{(lowpx_0 - py_0) \times (lowpx_0 - py_0)}{(lowpx_0 \times (1 - lowpx_0))/n_x + (py_0 \times (1 - py_0))/n_y}, \frac{(px_0 - lowpy_0) \times (px_0 - lowpy_0)}{(px_0 \times (1 - px_0))/n_x + (lowpy_0 \times (1 - lowpy_0))/n_y}\right)$
    // Continue only if enough haplotypes remain for test to find different distributions
    if $maxscore_0 \geq C$ then
        $score_0 \leftarrow max(score_0, compare(Sx_0, n_x, Sy_0, n_y, d + 1, D))$
        if $score_0 \geq C$ then
            return (similar = false, score = n/a)
    $lowpx_1 \leftarrow \alpha/(n_x + \alpha + \beta)$
    $lowpy_1 \leftarrow \alpha/(n_x + \alpha + \beta)$
    $maxscore_1 \leftarrow max\left(\frac{(lowpx_1 - py_1) \times (lowpx_1 - py_1)}{(lowpx_1 \times (1 - lowpx_1))/n_x + (py_1 \times (1 - py_1))/n_y}, \frac{(px_1 - lowpy_1) \times (px_1 - lowpy_1)}{(px_1 \times (1 - px_1))/n_x + (lowpy_1 \times (1 - lowpy_1))/n_y}\right)$
    if $maxscore_1 \geq C$ then
        $score_1 \leftarrow max(score_0, compare(Sx_1, n_x, Sy_1, n_y, d + 1, D))$
        if $score_1 \geq C$ then
            return (similar = false, score = n/a)
    // Finally, we can say the distributions are close enough to merge two nodes
    return (similar = true, score = max(score_0, score_1))
```



Algorithm 5 *A simple subroutine for splitting two sets of haplotypes Input is (i) a set of haplotypes X, all of length D and (ii) an allele position $1 \leq d \leq D$. Output is two subsets S_0 and S_1 such that all haplotypes in S_0 have a zero at allele d and all haplotypes in S_1 have a one at allele d*

```
procedure split $(X, d)$
    $S_0 \leftarrow$ ⊠
    $S_1 \leftarrow$ ⊠
    for each haplotype $h$ in $X$ do
        if $h_d = 0$ then // if the allele at position $d$ is zero...
            $S_0 \leftarrow S_0 \bigcup \{h\}$// add $h$ to $S_0$
        else
            $S_1 \leftarrow S_1 \bigcup \{h\}$
    return $S$// where $S$ is $\{S_0, S_1\}$
```

2.4 Evaluation

In order to evaluate our phasing method, we require samples with known phase. To get these data, we collect trios of genotype samples from the AncestryDNA database. These are groups of three genotypes where one is the father, one is the mother, and one is the child. With trio data, we can easily separate each parental sample's two chromosomes into the one that was inherited by the child and the one that was not. The phase is unambiguous at all sites except the sites where all three samples are heterozygous. In these cases, phasing algorithms must use a model (such as those described in the previous section) to decide the phase. To avoid this problem, in our evaluation we use only the unambiguous sites (i.e., involving at least one homozygous SNP) of trio-phased samples in our assessment of phasing accuracy against the trio-phased "gold standard". We retain about 93% of SNPs and about 81% of heterozygous SNPs.

We examine 1,188 unrelated (e.g., only the parents) trio-phased samples as a test set. To evaluate our phasing algorithm, we compare the phase that the algorithm produces to that of the test set, and count (i) phase errors (see Figure 2.4), the number of times that the phase disagrees among heterozygous sites that are unambiguous in the duo or trio, and (ii) impute error, the rate at which SNPs are incorrectly imputed when 1% of SNPs are set to missing in the input data.

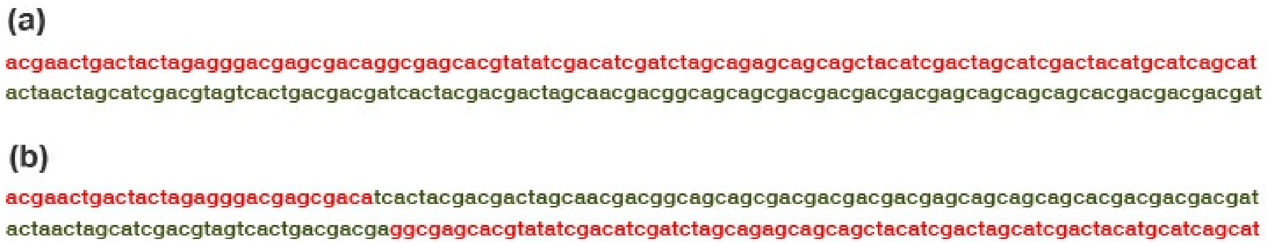

(a)
acgaactgactactagagggacgagcgacaggcgagcacgtatatcgacatcgatctagcagagcagcagctacatcgactagcatcgactacatgcatcagcat
actaactagcatcgacgtagtcactgacgacgatcactacgacgactagcaacgacggcagcagcgacgacgacgacgagcagcagcagcacgacgacgacgat
(b)
acgaactgactactagagggacgagcgacatcactacgacgactagcaacgacggcagcagcgacgacgacgacgagcagcagcagcacgacgacgacgat
actaactagcatcgacgtagtcactgacgacgaggcgagcacgtatatcgacatcgatctagcagagcagcagctacatcgactagcatcgactacatgcatcagcat

Figure 2.4: Example of a phasing error. (a) The true phase of a genotype sample. The haplotype inherited from one of the parents is colored red, and the haplotype inherited from the other parent is colored green. (b) The estimated phase of the same sample keeps the haplotypes mostly separate, but does contain an error.

We compare the performance of Underdog to that of Beagle [Browning AJHG 2007] and HAPI-UR [Williams AJHG 2012], another freely-available phasing algorithm (see Table 2.1 for the specific settings we use). We learn Underdog models from a set of 189,503 phased samples (which do not include any phased samples in the duos and trios used to create the test set) that were phased using HAPI-UR in sets of about 50,000 at a time. Beagle and HAPI-UR generally require many samples to phase together, and we experiment with various input set sizes (which always include the test set).

Program	Version	Settings
Beagle	3.3.2	nsamples=20, 10 iterations (which is default)
HAPI-UR	1.01	-w 84 (as is recommended by the manual) and default values for the other command-line parameters

Table 2.1: Experimental settings for the phasing algorithms we compare against, Beagle and HAPI-UR.

Our results are shown in Table 2.2. Because these phasing algorithms can phase all chromosomes in parallel, we have computed results using chromosome 1 only (specifically 52,129 SNPs on chromosome 1), which is approximately 8% of the genome.

Method	Supplementary Input Samples	Model Size	Total Phase Time	Phase Errors	Impute Error
Beagle	0	2,970,907	4h 14m 25s	2.60%	2.23%
Beagle	1,000	6,353,295	7h 9m 29s	2.09%	1.91%
Beagle	2,000	9,347,111	10h 16m 0s	1.90%	1.85%
Beagle	5,000	17,869,941	22h 41m 21s	1.63%	1.67%
HAPI-UR	0	N/A	2h 44m 17s	12.81%	33.43%
HAPI-UR	10,000	N/A	13h 13m 45s	2.00%	44.27%
HAPI-UR	20,000	N/A	1d 3h 13m 29s	1.59%	42.53%
HAPI-UR	50,000	N/A	3d 9h 7m 15s	1.23%	40.23%
Underdog	0 (Models are pre-computed)	102,692,825	5m 48s (4h 11m 8s on a single processor)	0.93%	1.09%

Table 2.2: Experimentally measured properties of various phasing methods on our proprietary test set (1,188 samples). Supplementary input samples are samples provided as input to the phasing program in addition to the test set. Model size for Beagle and Underdog models are the number of nodes in the union of all windows' Markov models. Total Phase Time is the phasing program's processing user time. Phase Errors (explained in the text) are the number of disagreements between duo/trio phasing and algorithmic phasing. Impute Error (explained in the text) is the rate at which the algorithm incorrectly replaces a missing genotype (based on artificially removing 1% of genotypes from input data).

The results in Table 2.2 show that Underdog succeeds in delivering high quality phasing, both in terms of a reduction in phasing error and imputation error. This is due primarily to our ability to build detailed phasing models using hundreds of thousands of samples. As the AncestryDNA database grows, we will be able to continually refine these models, likely with the effect of improved accuracy.

The results in Table 2.2 show that Underdog succeeds in delivering high quality phasing, both in terms of a reduction in phasing error and imputation error. This is due primarily to our ability to build detailed phasing models using hundreds of thousands of samples. As the AncestryDNA database grows, we will be able to continually refine these models, likely with the effect of improved accuracy.

Furthermore, by decoupling the construction of phasing models from the act of phasing samples, we also see a tremendous benefit in the computational cost for computing results for new samples. Only a one-time computational investment in model building is required, Similarly, since the models are static once built, there is no "batch effect" where phase of a sample could potentially be affected by the other samples with which it is phased. This ensures a more accurate interpretation for phased samples.

With Underdog, we have tackled the first challenge in accurate and efficient IBD analysis. In the next section, we introduce how we use this Underdog phased data to identify shared identical haplotypes between pairs of individuals.

3 Inferring IBD with the J-Germline Algorithm

3.1 Matching Algorithm

As discussed in Section 1, given phased genotype data for a number of samples, the task of finding genetic relatives begins with identifying segments of DNA that are identical between two people. That means that they have a haplotype in common. However, even the best phasing algorithms are not perfect. That means that any approach for identifying common haplotypes will always have to consider phasing error, or in other words, allow the identification of common haplotypes to disagree to some degree with the inferred haplotype phase (see Section 2, and Figure 2.4 for a description of a phasing error).

In addition to phasing error, AncestryDNA encounters a large and constantly-growing set of genotype samples. The number of comparisons required to compare each pair of samples in the database grows quadratically with database size, thus making a pairwise genotype alignment and scan intractable.

The approach we use to solve these two problems is based on the identity-by-descent (IBD) detection algorithm GERMLINE [Gusev GenRes 2009], which we summarize here. First, GERMLINE addresses computational speed concerns in part by using a hashing scheme to quickly identify short exact matches. Furthermore, GERMLINE allows for phasing error by extending these short exact matches. To see how it works, consider Figure 3.1 and the more detailed description below. After a sample is phased (see Section 2 and Figure 3.1A), GERMLINE breaks the genotypes up into windows (see Figure 3.1B and Table 3.1 for details). Since the genotypes are phased, each window consists of two strings of letters called haplotypes. For each sample s, haplotype h, and window w, GERMLINE computes f(h,w). The function f is called a hash function and it maps a character string and window identifier to an integer value. It has the property that if two different people have the same haplotype in the same window, they will have the same value of f(h,w). This makes it possible to quickly identify exact matches:

- Store all the values of f(h,w) computed for different samples in a table (called a hash table) of all possible integer values.
- For each new sample, lookup table entry f(h,w) and see if there are other samples in the table with the same haplotype.
- Once the short matches are discovered (Figure 3.1C), GERMLINE examines both genotypes in regions of the genome on both sides of the match to decide the extent of IBD in the region, allowing for phasing errors (i.e., switching between inferred haplotypes) until a SNP is reached where each individual is homozygous for different alleles (Figure 3.1D).
- Finally, after each match is extended, segments are filtered according to a pre-specified minimum size.

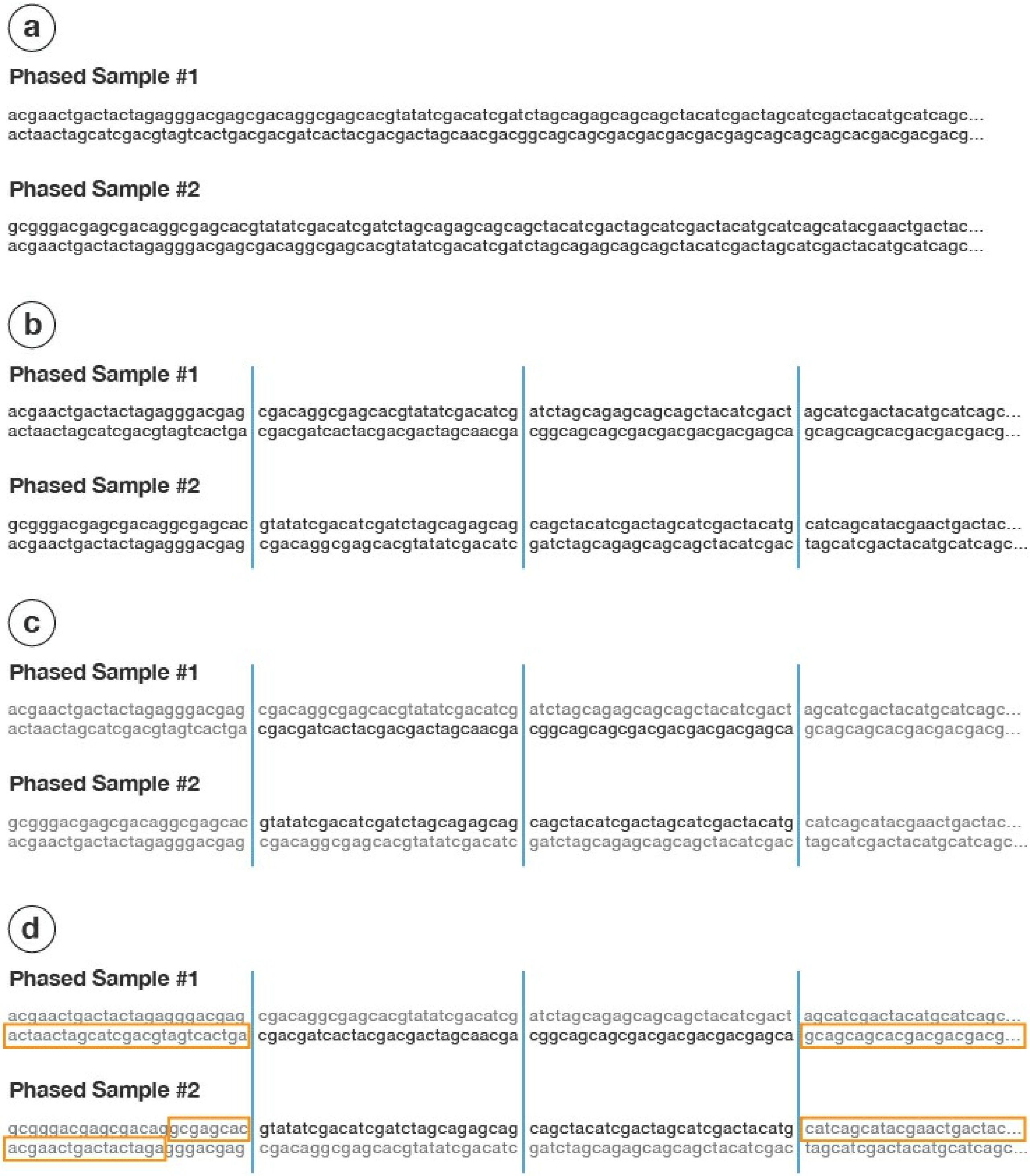

a
Phased Sample #1
acgaactgactactagagggacgagcgacaggcgagcacgtatatcgacatcgatctagcagagcagcagctacatcgactagcatcgactacatgcatcagc...
actaactagcatcgacgtagtcactgacgacgatcactacgacgactagcaacgacggcagcagcgacgacgacgacgagcagcagcagcacgacgacgacg...
Phased Sample #2
gcgggacgagcgacaggcgagcacgtatatcgacatcgatctagcagagcagcagctacatcgactagcatcgactacatgcatcagcatacgaactgactac...
acgaactgactactagagggacgagcgacaggcgagcacgtatatcgacatcgatctagcagagcagcagctacatcgactagcatcgactacatgcatcagc...
b
Phased Sample #1
acgaactgactactagagggacgag cgacaggcgagcacgtatatcgacatcg atctagcagagcagcagctacatcgact agcatcgactacatgcatcagc...
actaactagcatcgacgtagtcactga cgacgatcactacgacgactagcaacga cggcagcagcgacgacgacgacgagca gcagcagcacgacgacgacg...
Phased Sample #2
gcgggacgagcgacaggcgagcac gtatatcgacatcgatctagcagagcag cagctacatcgactagcatcgactacatg catcagcatacgaactgactac...
acgaactgactactagagggacgag cgacaggcgagcacgtatatcgacatc gatctagcagagcagcagctacatcgac tagcatcgactacatgcatcagc...
c
Phased Sample #1
acgaactgactactagagggacgag cgacaggcgagcacgtatatcgacatcg atctagcagagcagcagctacatcgact agcatcgactacatgcatcagc...
actaactagcatcgacgtagtcactga cgacgatcactacgacgactagcaacga cggcagcagcgacgacgacgacgagca gcagcagcacgacgacgacg...
Phased Sample #2
gcgggacgagcgacaggcgagcac gtatatcgacatcgatctagcagagcag cagctacatcgactagcatcgactacatg catcagcatacgaactgactac...
acgaactgactactagagggacgag cgacaggcgagcacgtatatcgacatc gatctagcagagcagcagctacatcgac tagcatcgactacatgcatcagc...
d
Phased Sample #1
acgaactgactactagagggacgag cgacaggcgagcacgtatatcgacatcg atctagcagagcagcagctacatcgact agcatcgactacatgcatcagc...
actaactagcatcgacgtagtcactga cgacgatcactacgacgactagcaacga cggcagcagcgacgacgacgacgagca gcagcagcacgacgacgacg...
Phased Sample #2
gcgggacgagcgacaggcgagcac gtatatcgacatcgatctagcagagcag cagctacatcgactagcatcgactacatg catcagcatacgaactgactac...
acgaactgactactagagggacgag cgacaggcgagcacgtatatcgacatc gatctagcagagcagcagctacatcgac tagcatcgactacatgcatcagc...

Figure 3.1: A) Phased genotype data for two people over a small part of the genome. B) The same data in A, broken up into short windows. C) The same data in figure B, highlighting exact matches between the two samples. D) The genotypes to the left and to the right continue to match, if we allow for phasing errors.

First, we break the genome into several parts, each a significant portion of the genome (typically about half of a chromosome, but less for larger chromosomes), and filter out a subset of SNPs, with a focus on avoiding low-reliability SNPs, centromeres, and portions of the genome with low-recombination rate. We break up the resulting segments into windows of about 100 SNPs each (see Figure 3.1B) for input to the hash function.

We have designed and implemented a proprietary version of GERMINE, called J-Germline, made scalable through incremental sample processing and use of a MapReduce framework relying on open source large data processing software Hadoop [Dean CommACM 2008, Apache 2005] and open source database software HBase [Apache 2010, Chang ACMTCS 2008]. The phased haplotype data remain in HBase between executions of J-Germline, which allows it to report matches between new and previously analyzed samples. Moreover, we implement J-Germline using Hadoop and HBase, creating a fast, robust system that is not limited to a single workstation process, and so can be made more efficient by adding computing resources.

3.3 Performance

To demonstrate the necessity of our specialized implementation, we run benchmarking experiments that compare the running times of GERMLINE and of J-Germline. In the first experiment, we add samples to an initially empty database. Figure 3.2 shows the resulting running time. We expect the running time of GERMLINE to be quadratic in the size of the database (because the number of pairs, which is proportional to the number of pairs with matches, is (n×(n-1))/2, or n2/2 - n/2) and for the running time of J-Germline to be linear in the size of the database, because samples already in the database do not need to be compared to each other. Figure 3.2 shows our results, which confirm our expectations. (To make the results comparable, we use only one process (one Hadoop mapper) to do the J-Germline computation.)

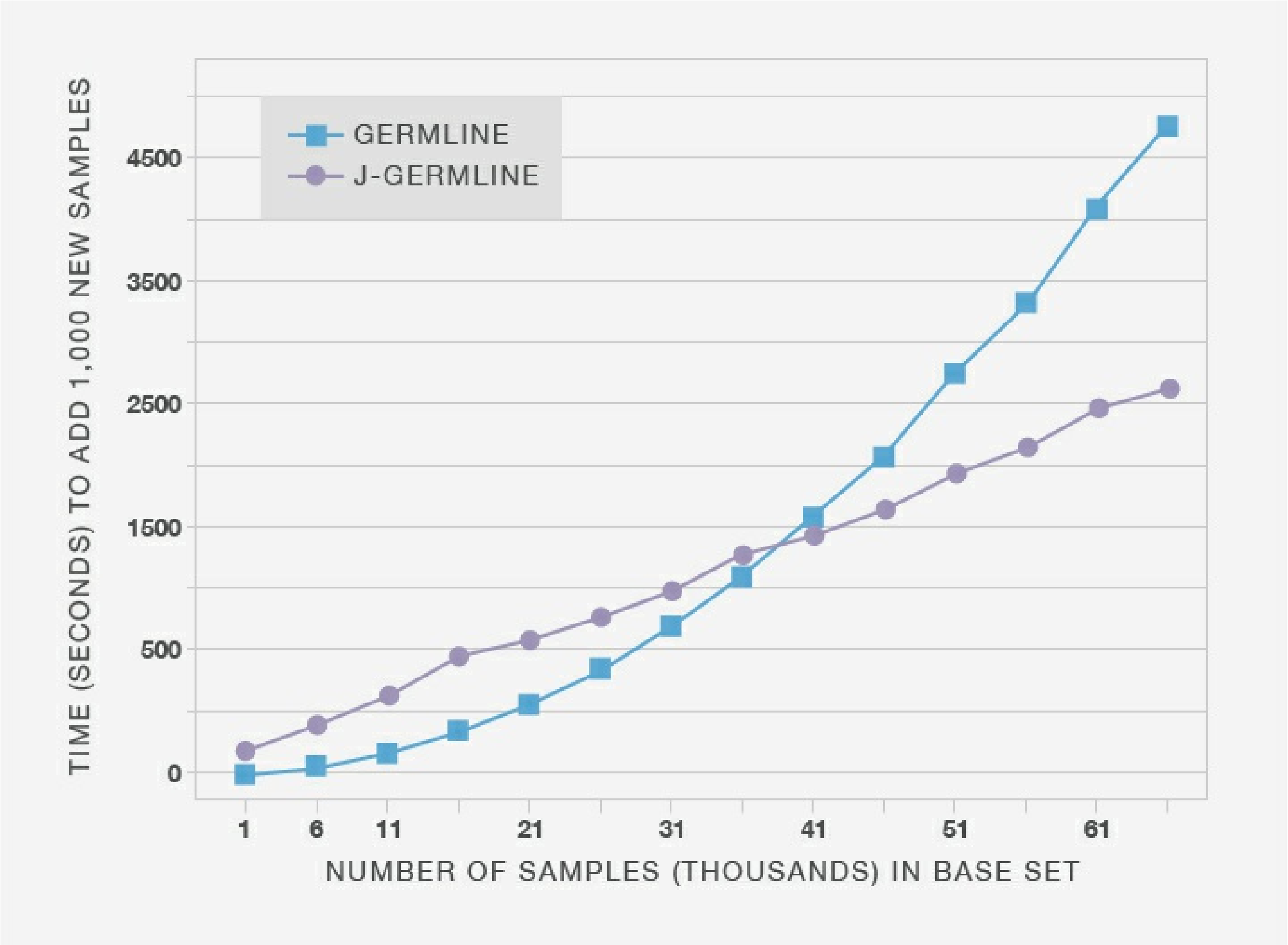


Figure 3.2: GERMLINE vs J-GERMLINE for incremental sample set growth. In the case of adding batches of 1,000 new samples incrementally, runtime grows linearly for J-GERMLINE (purple circles) and quadratically for GERMLINE (blue squares) when the data set grows incrementally. This is because GERMLINE must re-compute all results each run. To make the comparison fair we have required J-GERMLINE to use only a single mapper (thread).

The next experiment shows the advantage of adding resources to our distributed J-Germline approach.

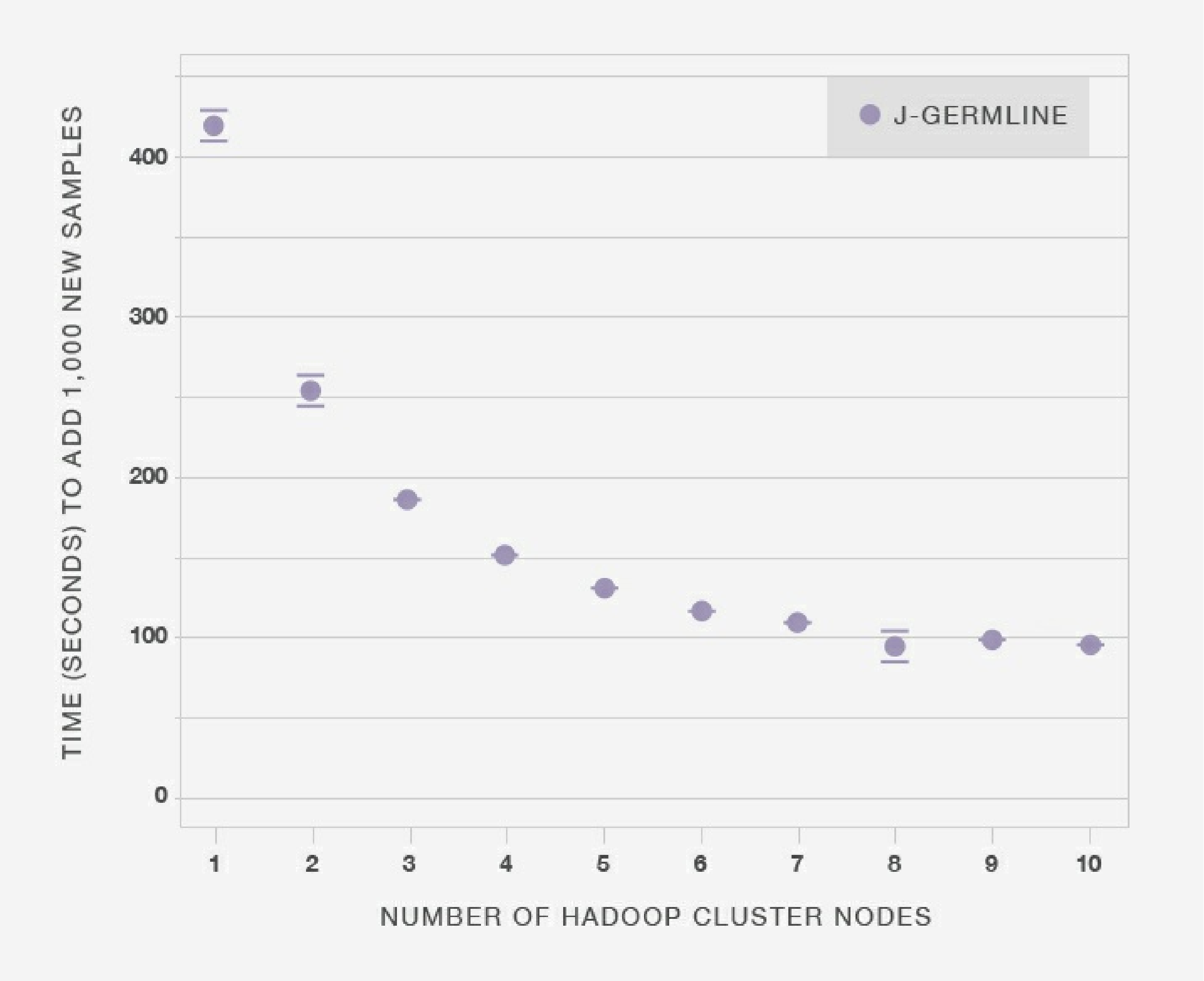


Figure 3.3. J-GERMLINE runtimes as a function of the number of nodes in a Hadoop cluster. To assess the effect of Hadoop cluster size on J-GERMLINE runtimes, we plot the results (mean ± standard error of ten replicates) of adding 1,000 new samples to a base set of 20,000 previously processed samples for different sizes of Hadoop cluster. In this experiment each node in the cluster runs up to 16 mappers simultaneously. Runtimes improve as the cluster grows until reaching a minimum runtime of 100 seconds with 6 nodes at which point additional nodes provide only small gain.

Figure 3.3 shows that adding additional computational nodes to a Hadoop cluster can greatly reduce J-GERMLINE runtimes. In the case of adding a new batch of 1,000 samples to a base of 20,000 previously analyzed samples, we reach a minimum runtime of approximately 100 seconds with six nodes.

J-GERMLINE overcomes our second large IBD analysis challenge -- it allows us to rapidly identify segments of DNA that are identical between two people in a very large, continuously growing database. By creating a parallelized implementation, we can continue to grow our computational infrastructure in response to future database growth. While identification of identical segments is a necessary part of IBD analysis, we must still measure the evidence that any shared segment is due to recent common ancestry, and thus meaningful for genealogy. We discuss this challenge and our novel solution in the following section.

4 Timber IBD Filtering Algorithm

The main goal of AncestryDNA IBD matching is to provide estimations of relationships between pairs of individuals that are meaningful for genealogical research. As a result, AncestryDNA is interested in finding pairs of individuals who share DNA identical-by-descent -- in that it was likely inherited from a recent common ancestor discoverable in genealogical time.

In Section 3, we discuss the method J-Germline that is used to identify segments of DNA that are identical between pairs of individuals. As we discuss in Section 1.3, however, there are a number of reasons why DNA between two individuals might be incorrectly observed to be identical, or possibly shared from very distant genealogical time. The objective of this section is to address how we study and filter matching segments inferred from J-Germline to yield a set of high-confidence inferred IBD segments that were likely to be inherited in two individuals from a recent common ancestor.

4.1 Motivation

In the output from J-Germline, we commonly observe certain short, specific locations of an individual's genome that match an excessive number of people. An example of this observation is shown for one individual in Figure 4.1. In this figure, the x-axis represents the windows across one chromosome and the y-axis represents the number of matches inferred with J-Germline for this individual in that window. This is an example "match profile."

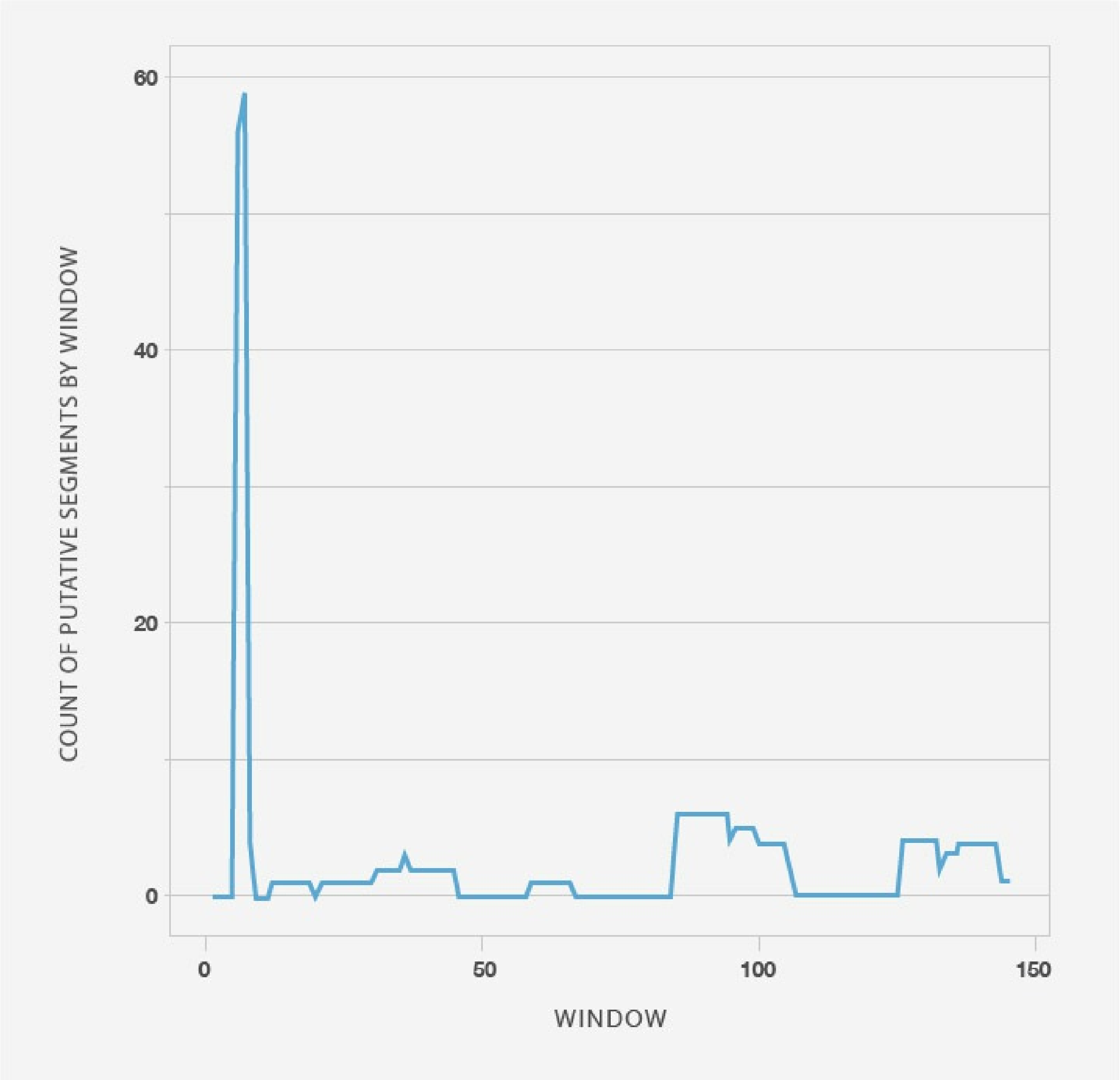


Figure 4.1: A section of the genome (in this plot approximately 14,000 SNPs) is divided into windows of 96 SNPs each. After computing the match results (using J-Germline) for an individual, we count the number of matches that an individual has with all other members across the database in each window. This plot shows that some windows have extreme numbers of matches for this particular individual.

Note that the pattern we observe in this match profile is not consistent with what we know about DNA that is actually shared IBD. If all of these matches were due to recent IBD, we would not expect to observe such an excessive variance in the numbers of matches (inferred with J-Germline) per genomic position. Instead, we expect a more constant rate of sharing across the genome. This leads to the natural conclusion that many of these "spikes" represent DNA that is not shared due to IBD from recent common ancestry, but for some other reason (see Section 1.4).

To further explain this point, consider Figure 4.2. Here, we show the histogram of the match counts for all the windows in the genome for the same individual. Once again, note that while most windows have a small number of matches, others have many times more.

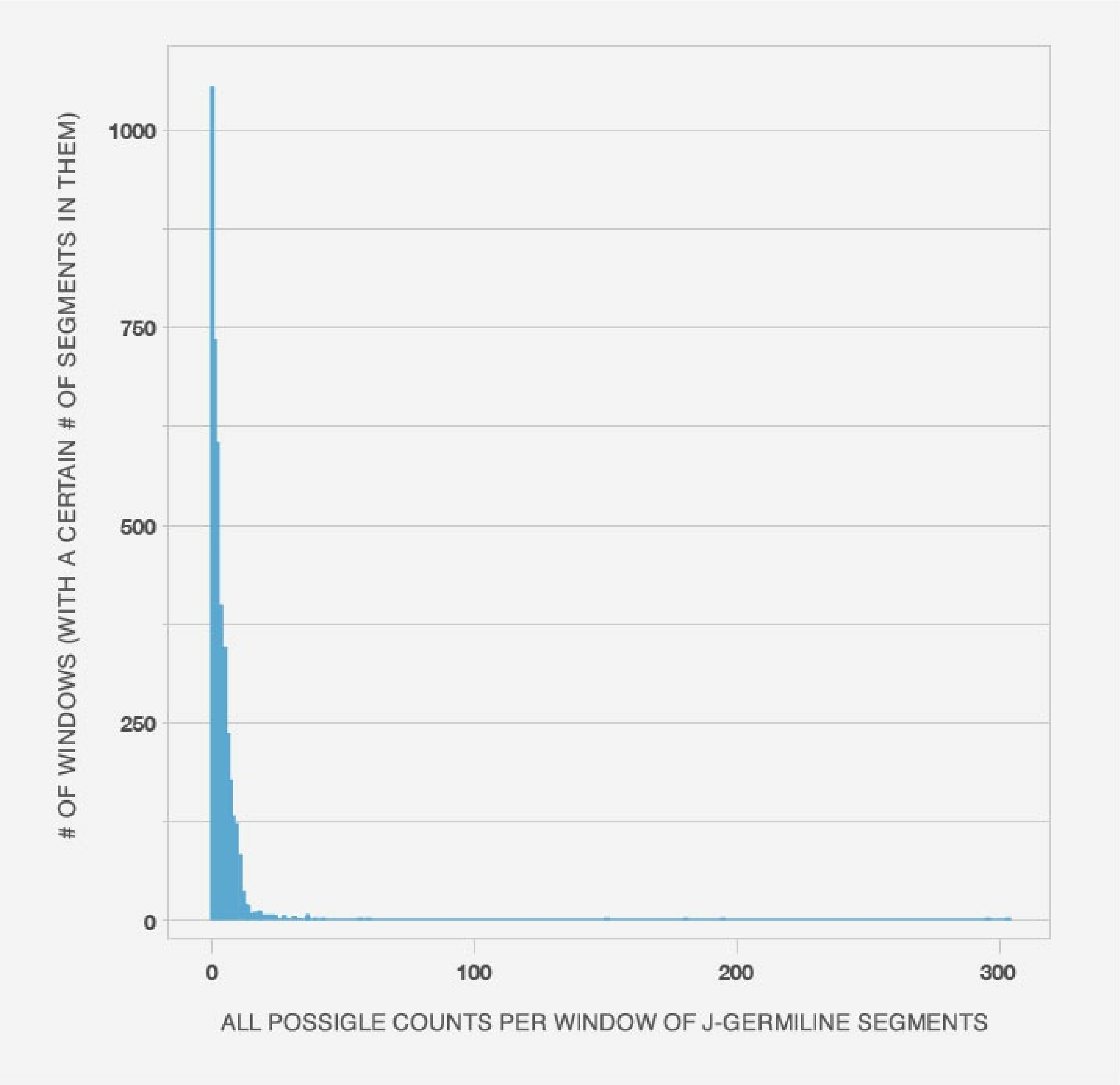


Figure 4.2: An example of the histogram of per-window match counts for all the windows in the genome for one individual.

On an individual basis, we can take advantage of this histogram to re-score each matched segment by down-weighting the windows that do not appear to convey information about recent genealogical history. In particular, we can weight a given matched window between two individuals based on the number of matches in that window in each individual -- relative to the number of matches across all genomic windows in each individual. This is the motivation behind the newly-developed method Timber.

4.2 Timber Algorithm

Our large database of genotype samples and matched segments (over 500,000 samples and over 2 billion matched segments between individuals at the time of writing this document) enables us to obtain insights from the individual specific match profiles shown above and gather statistics about which matched segments of the genome are likely to be false positives (i.e., the identical DNA is not evidence of IBD). The larger the number of matches for an individual, the better these statistical estimates can be. In order to compare all samples equally and avoid the computational complexities of a constantly-growing database, we use a fixed set of preselected samples to form a Timber reference set, R.

The steps of the procedure to compute a Timber score for a genotype sample S:

- Select a large set of genotyped samples as a reference set, R. Our reference set contains approximately 320,000 samples, which we find is sufficient to gather the matching statistics described below.
- Divide the genome into n short windows. We use the same windows that are used in the implementation of J-Germline.
- ount the number of matches calculated using J-Germline between sample S and R in each window in the genome. We denote the resulting histogram of counts (across windows 1 through n) CS = <CS,1,CS,2,...,CS,n> (for example, the one shown in Figure 4.2).
- Based on CS, compute the vector of weights WS = <WS,1,WS,2,...,WS,n> = f(CS), where each real-valued component WS,w is between 0 and 1 (i.e., 0 ≤ WS,w≤ 1, for all windows w). f is a proprietary optimization function based on a statistical analysis of the match profile CS.
- Store WS in the database (we use an HBase database in our implementation).
- For each match segment g involving S (and another individual, M, who may or may not be in the reference set), recompute the length of the segment as TimberScoreg =∑_(w∈g)h(length(w),WS,w , WM,w) , where h is a proprietary function that generally downweights the length of the windows in which S and/or M have a large number of matches.

Figure 4.3 formally illustrates the procedure, and Figure 4.4 illustrates the weight vector W computed for the individual shown in Figure 4.1. Finally, Figure 4.5 illustrates the new (post-Timber) match profile for the same individual. The general effect of Timber weights is to down-weight windows with a significantly higher-than-expected number of matches as they are shown to provide less evidence of IBD.

Algorithm 1 The Timber Procedure. Input will be the output from J-Germline, which is a list X of new (i.e., not already in the database) matching segments (individual 1 identifier, individual 2 identifier, segment start, segment end). f is a function to compute a vector of weights from a histogram of match counts. h is a function to reweight a window's interpreted length based on its actual length and two weights.

```
procedure timber (X)
    Let N ← the set of (new) individuals without weights
    Let R ← a large reference set already in the database

    for each i ⊠ N do // initialize per-window match counts $C_i$ to ⊠000...0⊠
        for each window w in the genome do
            $C_{i,w}$ ← 0// $C_{i,w}$ denotes the count in window w

    for each segment g in X indicating identical DNA between A and B do
        for each window w that overlaps g (even partially) do
            if A ⊠ R then // if A is in the reference set
                $C_{B,w}$ ← $C_{B,w}$ + 1// increment count
            if B ⊠ R then
                $C_{A,w}$ ← $C_{A,w}$ + 1

    for each i ⊠ N do // Let $W_i$ denote the weight vector for individual i
        Let $W_i$ ← f($C_i$)
        Store $W_i$ in database

    for each segment g in X indicating identical DNA between A and B do
        score ← ⊠$_{w⊠g}$ h(length (w), $W_{A,w}$ , $W_{B,w}$ )
        Attach score to segment g

    return g // return updated match segments
```

Figure 4.3: Pseudocode illustrating the Timber procedure.

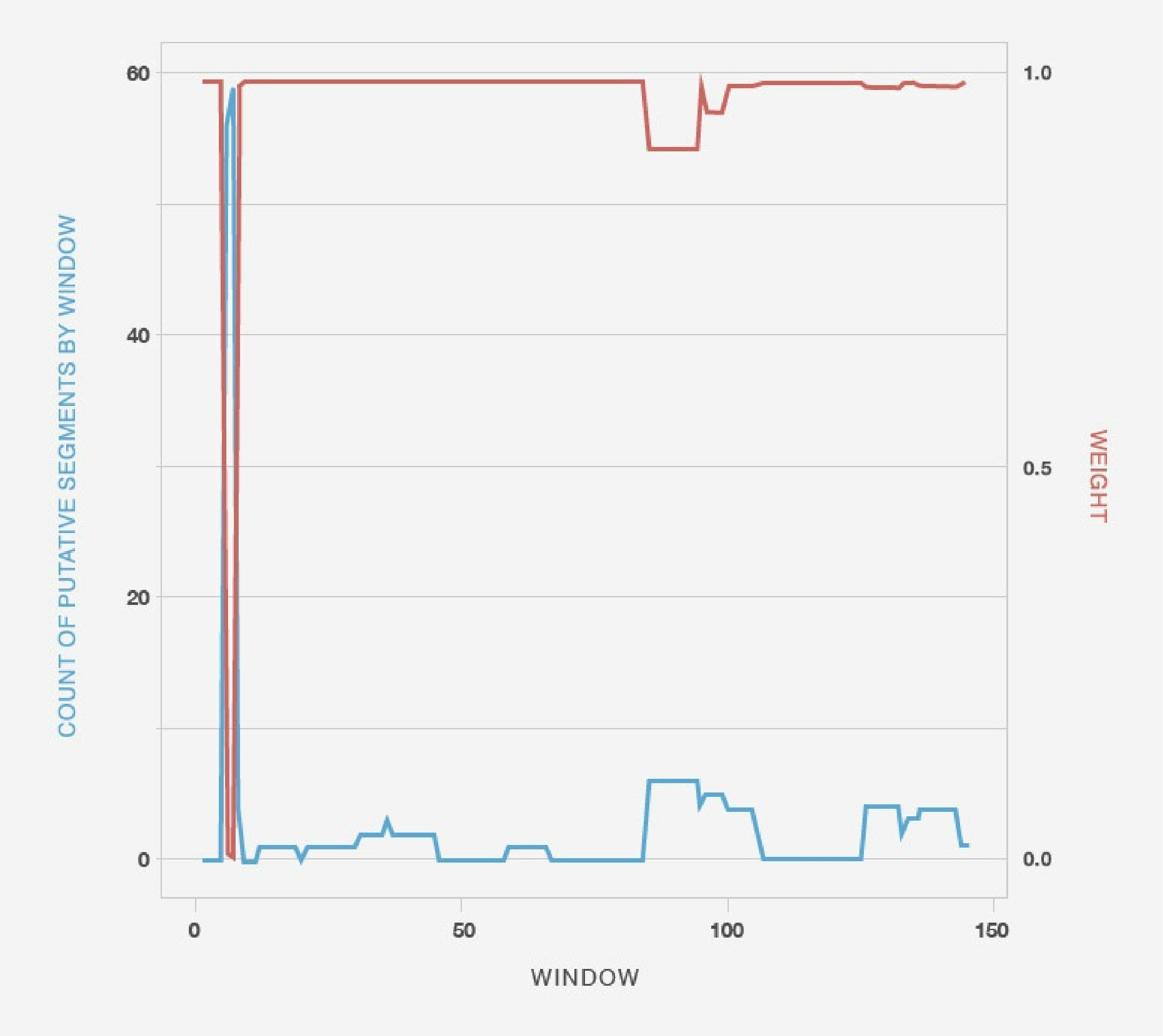


Figure 4.4: An example of per-window match counts on a chunk of the genome for one individual (in blue) along with the estimated weight for that window (in brown). For comparative purposes only, the weight is re-scaled so that a weight of 1 has a value of 59 according to the y-axis.

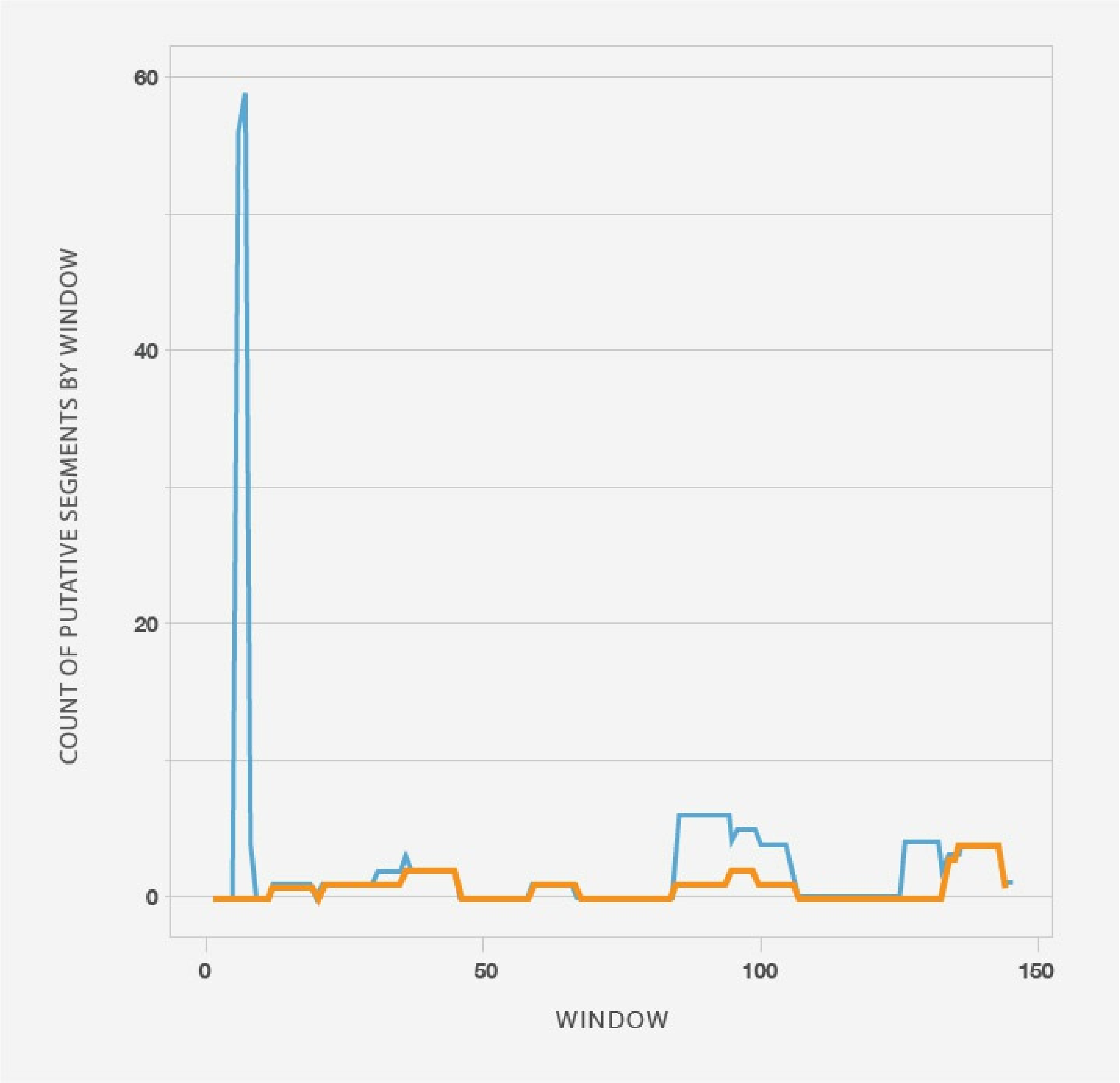


Figure 4.5: An example of per-window match counts on a chunk of the genome for one individual both pre-TIMBER (in blue) and post-TIMBER (in green).

Timber allows us to measure the evidence that each matched DNA segment is due to recent common ancestry. By taking advantage of our very large database of sample genotypes, we are able to use this evidence to, in an individual-specific manner, rescale each match based on the likelihood it is actually IBD. Through rigorous testing, we have found that Timber eliminates a large majority of false positive IBD segments. As we demonstrate in the following section, this significantly reduces the number of false positive relationships estimated between pairs of individuals in practice..

5 Relationship Estimation

5.1 Background

Once matching DNA segments are identified and analyzed by Timber, we interpret the matching segments for every pair of individuals. For a given pair of individuals, this means estimating the degree of relationship between them. In practice, determining this degree of the relationship is done by estimating the number of meioses separating the two individuals (see Figure 5.1).

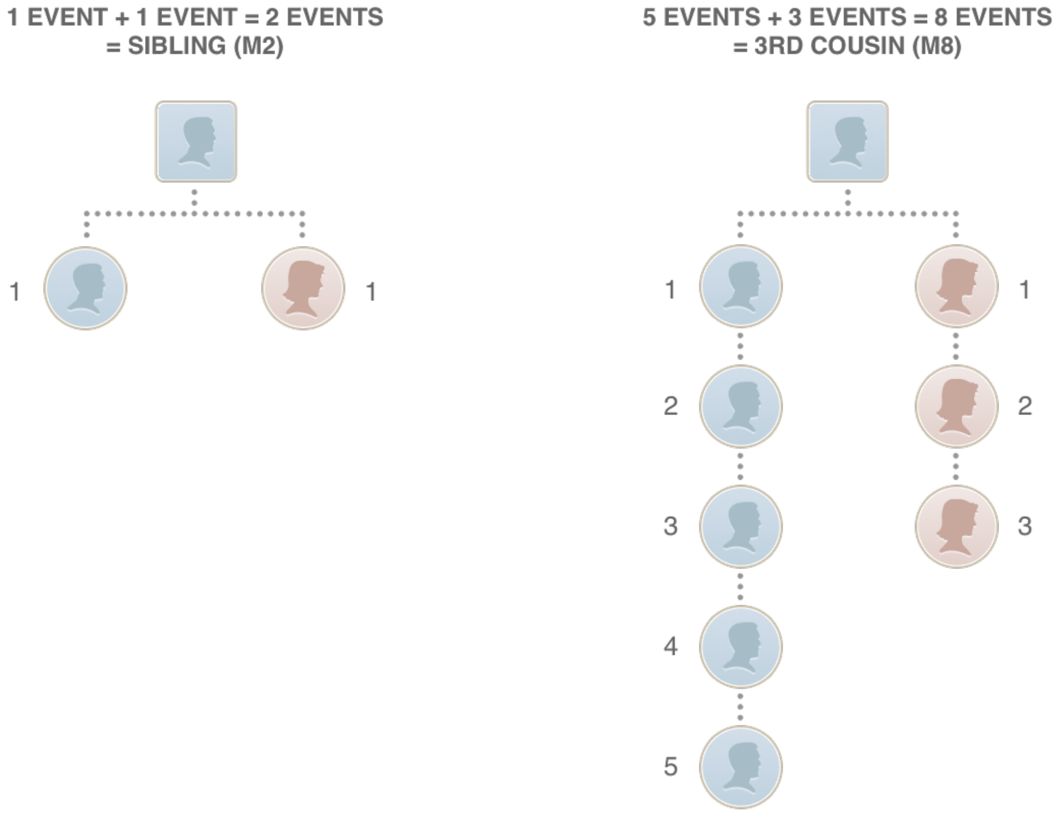


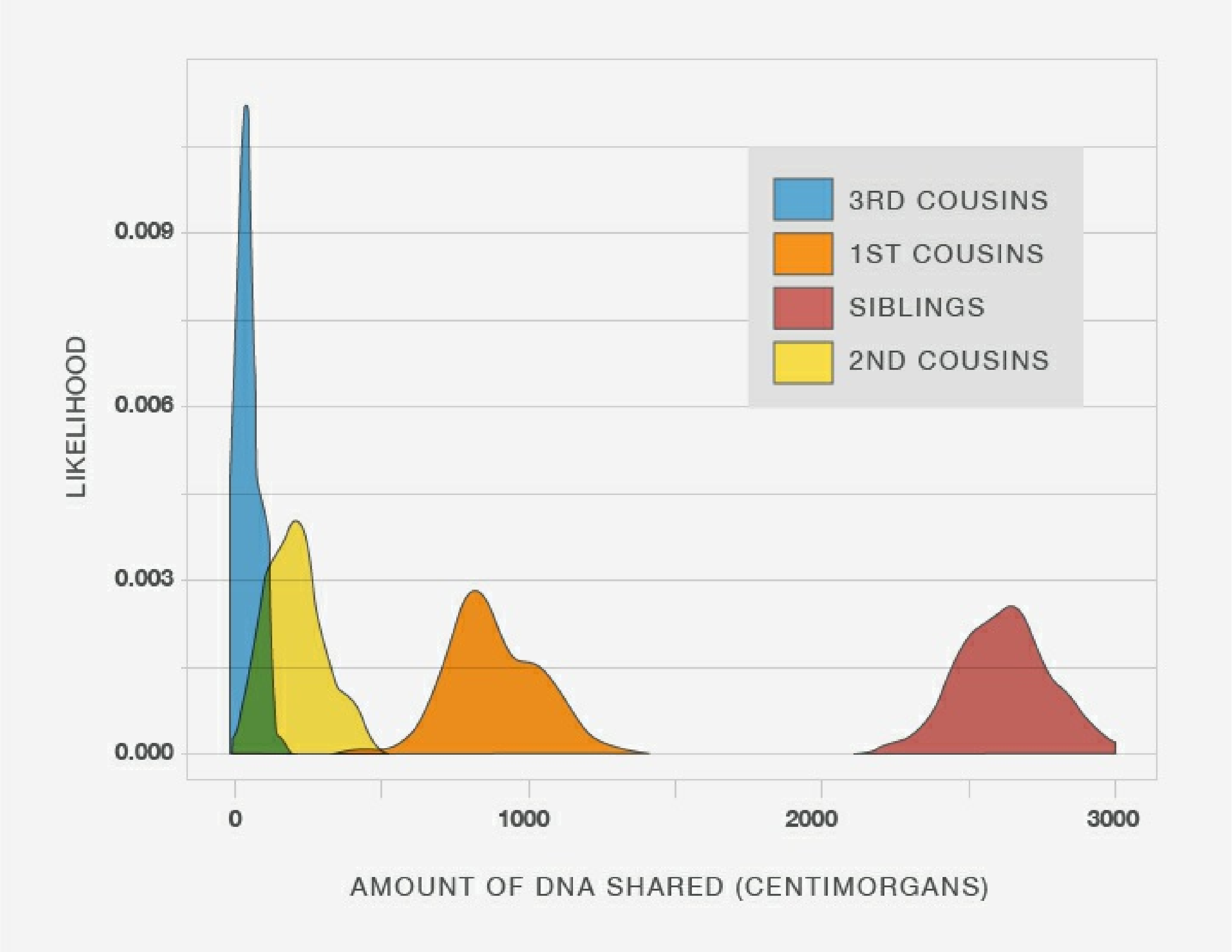
Figure 5.1. Figure demonstrating the number of meioses between individuals given their pedigree relationship. A) Demonstrates that two siblings are separated by two meioses -- one meiosis from the father to child1, and another from the father to child2. B) Demonstrates that two third cousins are separated by 8 meioses in this case: 5 from the common ancestor to the left-hand cousin, and 3 from the common ancestor to the right-hand cousin.

Referring back to Figure 1.1, a parent and child (separated by only one meiosis) will have identical DNA throughout the entire genome. Two siblings will have identical DNA throughout much of the genome, but since the recombination of DNA (before it is passed down from parent to child) is a random process, the amount of DNA that two siblings share IBD can vary. In general, the more generations removed from a common ancestor two related people are (i.e. - the more meioses separating their genotypes), the more uncertainty there is in the amount of DNA they will have both inherited from the common ancestor(s).

5.2 Relationship Estimation Method

In order to make the best estimate of the relationship between two individuals (e.g., the number of meioses separating them), we start by computing distributions of the amount of DNA that is expected to be shared IBD between individuals with various meiosis levels separating them. To do so, we apply our DNA matching algorithm (the combination of Underdog, J-Germline, and Timber) to a set of individuals with known relationships.

Then, we examined distributions of the amount of DNA shared IBD between each of these meiosis levels (an example of such distributions are shown in Figure 5.2). As expected, while distributions of the amount of IBD between individuals with low meiosis levels are rather distinct, distributions for more distant relationships (higher meiosis levels) overlap greatly. Figure 5.2 shows the distribution of amount of identical DNA for various relationships obtained from test data.


LIKELIHOOD
0.009
0.006
0.003
0.000
3RD COUSINS
1ST COUSINS
SIBLINGS
2ND COUSINS
0
1000
2000
3000
AMOUNT OF DNA SHARED (CENTIMORGANS)

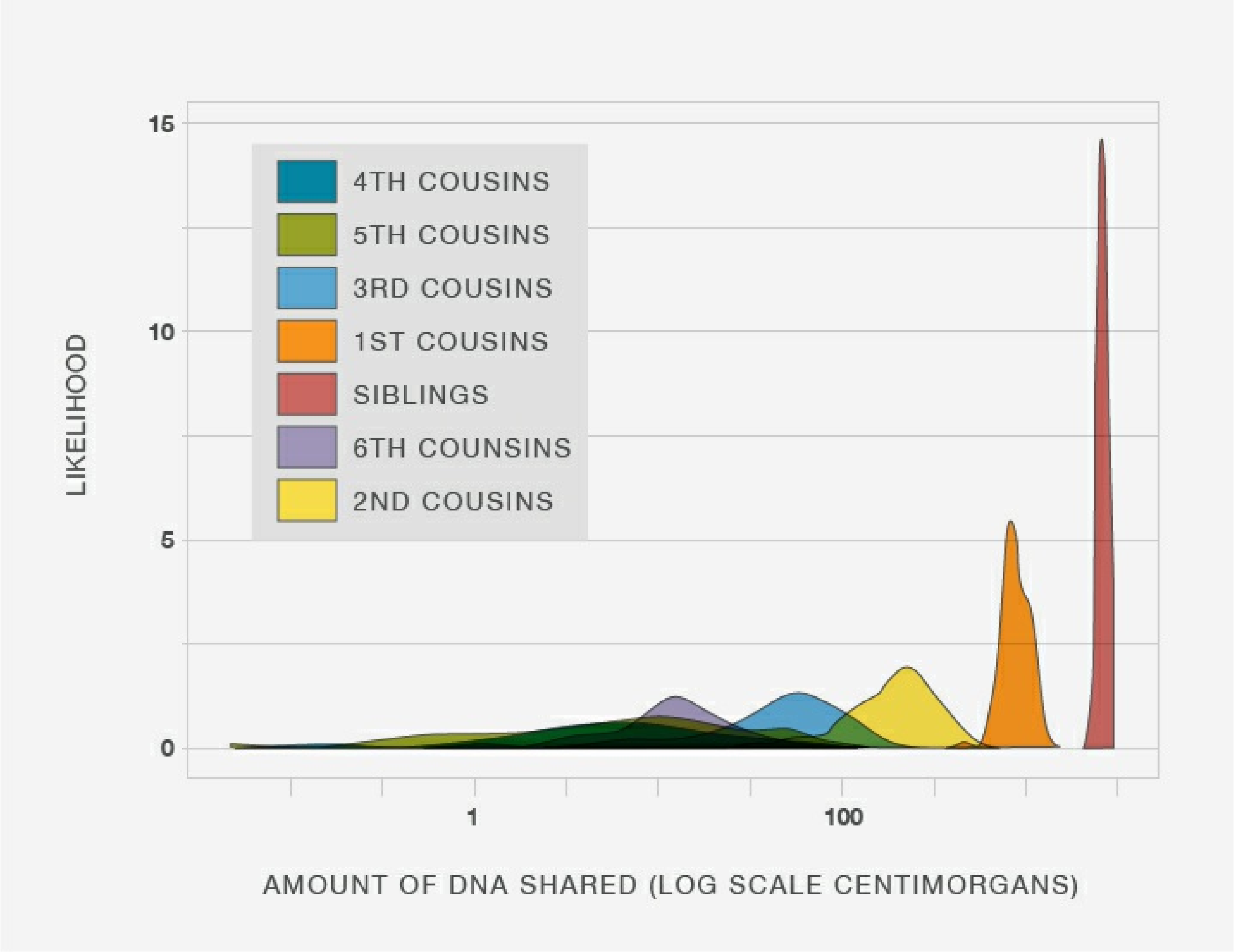


Figure 5.2: Probability density functions over the amount of sharing for various relationships. (top:) Siblings almost always share much more DNA than (e.g.,) first cousins or more distant cousins, but 2nd and 3rd cousins might (and are are fairly likely to) share the same amount, making them difficult to distinguish. (bottom:) To see more distant relationships as well, we put the horizontal axis on a logarithmic scale. These are the same data as the top figure, with 4th-6th cousins added in.

Then, we create a proprietary function that maps a set of IBD segments (and their lengths), each identified by J-Germline and adjusted by Timber, to a likelihood distribution over the possible relationships for a pair of individuals (i.e., number of meioses separating the two individuals). This function is chosen to be the one that best agrees with the relationship data described above.

In addition, we learn another function that maps a set of Timber-annotated segments to the likelihood that the two individuals involved in each match are related at all (through the past nine or fewer generations). This is because it is possible to share DNA IBD from a common ancestor further back than that, or to appear to share DNA IBD when there may not be a common ancestor at all. Therefore, for each match, we estimate not only the most likely relationship (as described above) but also the likelihood that there is a relationship at all.

5.3 Relationship Estimation Evaluation

To evaluate our relationship estimates, we start with a large data set of individuals with known relationships. We then compare the relationship estimate (e.g., "parent/child," "third cousin," or "unrelated") to the true relationship. Recall is the proportion of genuine IBD relationships that we are able to identify, and we define it for an individual as:

$$Recall = \frac{\text{Number of matches reported to the individual to whom the individual is genetically related}}{\text{Number of other individuals in the database to whom the individual is genetically related}}$$

Our test set contains over 150 genotyped samples from a large family with a well-researched pedigree containing about 2,500 relationships that vary from 1 meiosis to 15 meioses. In order to estimate recall, we must know whether a given pair of individuals has IBD. If we consider relationships more distant than third cousins, we cannot be certain that the pair of individuals share any DNA (third cousins, in fact, are only about 98% likely to have any IBD, let alone a detectable amount). For this reason, we cannot use this test set to estimate recall for distant relationships. Our recall rates on this test set are as follows:

Relationship	Estimated Recall
1. parent/child	100%
2. sibling	100%
3. aunt/uncle	100%
4. first cousin	100%
5. first cousin once removed	100%
6. second cousin	99.7%

Table 5.1 Estimated recall on large family with well-researched pedigree

However, we can also measure recall by simulating meiosis in a computer. This way, we can create a test set that has a specific amount of known IBD, even for more distant relationships. Our synthetic test set consists of over 7,000 samples, and over 300 of each of the relationships from parent/child to third cousin once removed. All relationships are simulated independently of all others. We estimate that, when a simulated relationship contains 5 cM or more IBD, our recall rates break down as follows:

Relationship	Estimated Recall
1. parent/child	100%
2. sibling	100%
3. aunt/uncle	100%
4. first cousin	100%
5. first cousin once removed	100%
6. second cousin	100%
7. second cousin once removed	98.7%
8. third cousin	95.9%

Relationship	Estimated Recall
9. third cousin once removed	90.9%

Table 5.2 Estimated recall on simulated data

For example, we estimate that, if an individual has a third cousin (row 8 in the table) with whom they share at least 5cM of IBD, there is about a 95.9% chance that we will detect it.

Another way we measure accuracy is by measuring the extent to which our estimates are correct. We define precision for an individual as:

$$Precision = \frac{\begin{array}{l}Number\ of\ \ matches\ reported\ to\ the\ individual\ to\\ whom\ the\ individual\ is\ genetically\ related\end{array}}{Number\ of\ matches\ reported\ to\ the\ individual}$$

We do not measure precision on our genotyped family data set, because we have no way of knowing about undocumented distant relationships in that family. However, on our synthetic test set, we know all of the genuine relationships. We estimate that the precision of our IBD analysis breaks down as follows:

Relationship	Estimated Recall
1. parent/child	100%
2. sibling	100%
3. aunt/uncle	100%
4. first cousin	100%
5. first cousin once removed	100%
6. second cousin	100%
7. second cousin once removed	100%
8. third cousin	95.3%
9. third cousin once removed	88.3%

Table 5.3. Estimated precision for simulated data

For example, if we estimate that an individual has a second cousin once removed (row 7 in the table), then there is about a 100% chance that the two individuals are actually related. Note however, that this table is not saying there is 100% chance that the two individuals are second cousins once removed. As you can see from the Figure 5.2, the amount of IBD between distant cousins varies, and a certain amount of IBD is a possible amount for potentially a few different relationships, but we report the most likely relationship.

In all, these results demonstrate the high precision and recall of the AncestryDNA matching algorithms.

6 Conclusions and Future Refinements

Our goal is to give AncestryDNA customers the best and most accurate information possible to help them in their genealogical research. The updates that we have described in this paper represent a major step forward both in the accuracy and richness of the information we provide.

In this paper, we have described a number of modifications and improvements to the AncestryDNA IBD estimation algorithm. These improvements have been made possible in large part due to the huge amount of data, including both genotypes and pedigrees, from the AncestryDNA community. Clearly, this data has been the means leading to the development of both Underdog and Timber, both of which require and were developed using large sets of data. Extensive testing has revealed that the various algorithms developed for performing IBD estimation and described here not only produce highly accurate estimates of IBD segments and relationships, but also are scalable to meet the computational demands of a continuously growing database.

While the current improvements that have been made are impressive, we expect that increasing amounts of data will enable us to continue to learn and improve our ability to phase, identify IBD, and deliver quality IBD matches to AncestryDNA customers. Not only may the action of algorithms such as Underdog and Timber be improved with increasing amounts of data, but additional IBD data across over a half-million individuals has the potential to yield fascinating new population genetic discoveries related to IBD estimation. Each match has the potential to lead a customer to that next genealogical discovery.

References

[Apache 2005] Apache. Welcome to Apache™ Hadoop®, 2005. http://hadoop.apache.org.

[Apache 2010] Apache. Welcome to Apache HBase™., 2010. http://hbase.apache.org.

[Browning AJHG 2011] B. L. Browning and S. R. Browning. A fast, powerful method for detecting identity by descent. The American Journal of Human Genetics, 88(2):173–182, 2011.

[Browning Genetics 2013] B. L. Browning and S. R. Browning. Improving the accuracy and efficiency of identity-by-descent detection in population data. Genetics, 194(2):459â€"–471, 2013.

[Browning AJHG 2006] Sharon R. Browning. Multilocus associate mapping using variable-length Markov chains. American Journal of Human Genetics, 78:903–913, 2006.

[Browning AJHG 2007] 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. American Journal of Human Genetics, 81:1084–1096, 2007.

[Browning NatGen 2011] Sharon R. Browning and Brian L. Browning. Haplotype Phasing: Existing Methods and New Developments. Nature Genetics 12:703-14, 2011.

[Chang ACMTCS 2008] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems, 26(2):4, 2008.

[Dean CommACM 2008] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[Durbin Nature 2010] R. M. Durbin, G. R. Abecasis, R. M. Altshuler, G. A. R. Auton, D. R. Brooks, A. Durbin, A. G. Gibbs, F. S. Hurles, F. M. McVean, P. Donnelly, M. Egholm, P. Flicek, S. B. Gabriel, R. A. Gibbs, B. M. Knoppers, E. S. Lander, H. Lehrach, E. R. Mardis, G. A. McVean, D. A. Nickerson, L. Peltonen, A. J. Schafer, S. T. Sherry, J. Wang, R. K. Wilson, R. A. Gibbs, D. Deiros, M. Metzker, D. Muzny, and J. Reid. A map of human genome variation from population-scale sequencing. Nature, 467(7319):1061–1073, 2010.

[Gusev GenRes 2009] A. Gusev, J. K. Lowe, M. Stoffel, M. J. Daly, D. Altshuler, J. L. Breslow, J. M. Friedman, and I. Pe'er. Whole population, genome-wide mapping of hidden relatedness. Genome Research, 19(2):318-26, 2009.

[HapMap Nature 2007] The International HapMap Consortium. A second generation human haplotype map of over 3.1 million SNPs. Nature 449, 851-861, 2007.

[Nelder CompJ 1965] J. A. Nelder and R. Mead. A simplex algorithm for function minimization. Computer Journal, 7(4):308–313, 1965. [Noto ASHG 2014] Keith Noto, Yong Wang, Ross Curtis, Julie Granka, Mat Barber, Jake Byrnes, Natalie Myres, Peter Carbonetto, Amir Kermany, Christine Han, Catherine A. Ball, Kenneth G. Chahine. Underdog: A Fully-Supervised Phasing Algorithm that Learns from Hundreds of Thousands of Samples and Phases in Minutes. Invited Talk, American Society on Human Genetics Annual Meeting, 2014.

[Purcell 2007] S. Purcell, B. Neale, K. Todd-Brown, L. Thomas, M. A. R. Ferreira, D. Bender, J. Maller, P. Sklar, P. I. W. De Bakker, M. J. Daly, and P. C. Sham. PLINK: a toolset for whole-genome association and population-based linkage analysis, 2007. http://pngu.mgh.harvard.edu/purcell/plink.

[Rabiner IEEE 1989] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2):257–286, 1989.

[Ron JCompSysSci 1998] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. J. Comp Syst. Sci., 56:133–152, 1998.

[Welch Biometrika 1947] B. L. Welch. The generalization of "Student's" problem when several different population variances are involved. Biometrika, 34(1-2):28–35, 1947.

[Williams AJHG 2012] A. L. Williams AL, N. Patterson, J. Glessner, H. Hakonarson, and D. Reich. Phasing of many thousands of genotyped samples. American Journal of Human Genetics, 91:238–251, 2012.