# HapROSE
# A Software Package for
# Long Range Haplotype Phasing

Samuel Angelo Crisanto
Honors Thesis in Computational Biology

April 2016

*Dedicated to my Father*

# Contents

# Abstract

HapROSE is a software package for Long Range Haplotype Phasing that implements different heuristics for haplotype clustering. I examine how a model of variable-memory inference proposed by Ron et. al. (1996) compares with the output of a variant which is at the heart of BEAGLE (Browning and Browning 2006). The BEAGLE heuristic adapted the algorithm proposed in the Ron et. al. paper in order to optimize for speed and a finite number of training samples, and is a current gold standard algorithm for this problem. I then implemented the stochastic EM algorithm used by BEAGLE, and compare phasing accuracy between the two heuristics under a number of different conditions. I also explore new methods of inferring the existence of haplotype blocks through the use of tracts in long reads, leveraging state of the art PacBio genome sequencing data and the Positional Burrows Wheeler transform.

# 1 Introduction

Humans are diploid organisms: their cells contain two sets of chromosomes, one from each parent. The DNA sequence of one of these chromosomes is called a haplotype. A genotype is the set of both haplotypes. Haplotype phasing is the problem of inferring a likely set of haplotypes that, when combined, produce an observed genotype. This becomes an issue because modern methods of genome sequencing require that the DNA be cut into smaller pieces, and reassembled from their overlapping segments. This works for a majority of the DNA strand since large portions of it are identical and will agree, but where there are mutations there is potential for half of the fragments which map to a location to have evidence of one variant, and half of the fragments to have evidence of a different variant. This ambiguity makes it difficult to perform inference tasks, since there is a large combination of possible haplotypes that could have generated any particular observed genotype. While this phasing problem can be solved exactly for shorter sequences, it becomes computationally intractable at the chromosome scale, leading to a rich variety of proposed heuristics. HapROSE implements an algorithm for variable-memory inference put forward by Ron et. al. [Ron et al., 1995] and compares its performance with that of a modification of the algorithm proposed by Browning and Browning (2006), and implemented in their BEAGLE [Browning and Browning, 2007a] software package. This BEAGLE algorithm is a modified form of Ron's that changes the criteria for node similarity to be a function of empirical counts. I investigate the ways in which this different behavior affects how the structure of the model being used to perform phase inference differs from that of a model constructed using Ron's algorithm, which is proven to converge asymptotically to the true distribution.

This paper also explores new methods of identifying haplotype blocks through the use of tracts in long reads, and proposes an application for state of the art PacBio genome sequencing data and the Positional Burrows Wheeler transform. We discuss how HapROSE can be enhanced to be used alongside a set of algorithms developed at the Istrail Lab collectively called HapCompass, a tool for phased haplotype assembly. We discuss how this additional support would enable the algorithm to accurately perform a long-range (chromosome wide) phasing while minimizing any switch errors (wrongful assignments of a phasing to a chromosome) that would occur as the algorithm transitioned between haplotype blocks.

# 2 The Haplotype Phasing Problem

DNA is a polymer of four nucleotide base pairs Adenine, Cytosine, Thymine, and Guanine, commonly abbreviated A, C, T, and G respectively. A human being has two different copies of each chromosome, and the DNA sequence of one of these strands is called a haplotype. We can treat the sequence of bases in each haplotype as a long string from the four-letter alphabet $\{A, C, T, G\}$. The

set of two haplotypes present in a human is called that human's genotype. It is of great biological interest to identify those places where the DNA sequences of two humans differ, since DNA serves as a "blueprint" for proteins in the human body. Many diseases that involve the incorrect folding, malfunctioning, or misregulation of a protein can trace the defect back to some variation of DNA in that organism's genotype. When performing inference to determine which differences are of interest, it is often more beneficial to consider only those positions in the DNA strand where there are known different variations from person to person. We call these differences "Single Nucleotide Polymorphisms", or SNPs.

To make this concept more concrete, let us consider a particular example. Imagine two strands of DNA $N$ bases long that are mostly identical in much of the human population except for at two positions, $i$ and $j$, $0 \leq i \leq j \leq N$, where WLOG at position $i$ some percent of the population has a $C$ and the remainder has a $G$, and at $j$ some percent of the population has a $T$ and the remainder has a $G$. We note that we do not need the entire DNA sequence from 0 to $N$ to uniquely identify which haplotype an individual possesses - we merely need to know which variant they have at position $i$ and which variant they have at position $j$. We make another simplifying assumption when defining the haplotype phasing problem - the *infinite sites* assumption. This assumes that mutations are uniformly distributed along the genome, and since there are so many base pairs ($\tilde{3}.2$ billion) in the human genome we assume that a mutation can occur exactly once at any given site. This results directly in the biallelic assumption, which is that at any given site $i$ there are only two different possible variants, or "alleles" which we will arbitrarily encode as 0 and 1.

This allows us to describe a haplotype of size $N$ containing $n \leq N$ variant sites as a sequence $\{0,1\}^n$, a more convenient and usually much shorter string. Let us then define a genotype, which the reader may recall is a set of two haplotypes, as a sequence $\{0,1,2\}^n$ where

- a 0 is placed at position $i$ if both haplotypes that compose the genotype have a 0 at that position
- a 1 is placed at position $i$ if both haplotypes that compose the genotype have a 1 at that position
- a 2 is placed at position $i$ if one haplotype has a 0 at position $i$ and the other haplotype has a 1 at position $i$

We define the above process of generating a genotype from two haplotypes as *conflating* two haplotypes, and will use $+$ to denote conflation. We will use the notation of Halldorsson [Halldorsson et al., 2004] and write $h + \overline{h} = g$ if the conflation of $h$ and $\overline{h}$ is $g$. In addition, we will use the kronecker delta $\delta_{h+\overline{h}^g} = 1$ if $h + \overline{h} = g$, and 0 otherwise. Finally, we will use $\phi_h$ to denote the probability of a haplotype, and $\phi_g$ to denote the probability of a genotype. As coined by Gusfield [Gusfield, 2000], we will say that a site is ambiguous in a genotype if it has a value of 2.

## 2.1 Haplotype Phasing

We formally define the problem of haplotype phasing as follows:

- **Given**: $G$, an $n \times m$ matrix of genotypes
- **Output**: $H$, a $2n \times m$ matrix of haplotypes that "explains" $G$, i.e.:

$$\forall g \in G \; \exists h_1, h_2 \in H : h_1 + h_2 = g$$

or, equivalently,

$$h_{2i} + h_{2i+1} = g_i, \; i = 0, 1, 2, \ldots, m$$

The phasing problem is complicated by the fact that we do not just seek any set of haplotypes H that adequately explains $G$, but rather we seek the set of haplotypes that correspond to the true haplotypes of the person whose genotype we are performing inference on. We note that if we simply go from left to right on a genotype and "flip a coin" to assign a 0 or 1 to any ambigouous site to create a single explanatory haplotype, the other haplotype is fully determined. However, the probability of this explanation being biologically correct under the assumption that all haplotypes are equally likely is $\frac{2}{2^k}$, where $k$ is the number of ambiguous sites, since we could have arbitrarily generated either of the haplotypes that, when conflated, equal that genotype. This is a very low probability, so we must appeal to a number of features intrinsic to the underlying biology that allow us to determine whether a phasing is "good" by assuming certain properties hold. Doing so allows us to gauge our success by means of some objective function.

Trivially, an ideal objective function would be to use the edit distance between the inferred haplotypes and the true haplotypes, but in most cases this inference is being performed precisely because the true haplotypes are unknown. What follows is a brief review of some objective functions for evaluating the quality of a phasing - readers wishing to dive deeper into the subject are invited to read [Halldorsson et al., 2004]

### 2.1.1 Parsimony Phasing

The principle of *parsimony* states that a phasing is "good" if it uses the fewest possible different haplotypes to explain the genotypes. That is, the set of distinct haplotypes in $H$ is as small as possible. This principle stems from the assumption that individuals in a population share a common ancestor, so it is more likely for them to have shared haplotypes than for them to have different haplotypes.

### 2.1.2 Maximum Likelihood Phasing

Maximum Likelihood phasing seeks to find the most likely set of haplotypes that could have been conflated to explained the observed genotypes. This assumes Hardy-Weinberg Equilibrium, which is to say, the probability of observing a genotype is equal to the product of the probabilities of observing its haplotypes. More formally,

$$\phi^g = \sum_{h+\overline{h}=g} \phi_h \phi_{\overline{h}}$$

Under this framework, we seek a set of haplotypes $H$ that maximizes

$$L(\phi_H) = \prod_{g \in G} (\phi^g)^n \hat{\phi}^g$$

subject to the constraints

$$\sum_{h \in H \phi_h} = 1 \text{ and } \phi_h \geq 0 \forall h \in H$$

## 2.2 Short Range Phasing

We say that a phasing problem is "short range" if there are 100 or fewer sites being considered. For a short range phasing problem, there are typically few enough possible solutions that we can practically enumerate them in the exponential amount of space and time required. Oftentimes for these sorts of problems we can optimize the above objective functions exactly. There are a variety of methods for short range phasing: the two most relevant to our purposes are explained in the following sections.

### 2.2.1 Clark Phasing

The underlying principle of Clark phasing is simple: if we have seen a haplotype before, it is more likely to occur again in a sample than a haplotype which we have not seen before. We iteratively apply a rule called "Clark's Rule", which states that we should try to resolve a genotype with a haplotype we have seen or inferred the existence of before. The general algorithm proceeds as follows:

There are a number of weaknesses associated with Clark phasing - if we cannot find any genotypes that can be unambiguously phased, we cannot begin the algorithm. Likewise, since it is taking a greedy approach to phasing, different permutations of haplotypes will result in different phasings depending on which haplotypes are resolved before others. Finally, if any genotypes are left unexplained after this process, they are said to be "orphaned", and it is unclear how to try to explain them.

A brute force method to find the optimal set of haplotypes to explain a set of genotypes under the clark rule would be to perform the above algorithm on ever permutation of the haplotypes. By enumerating all permutations, we

**Algorithm 1** Clark Phasing

**for** each genotype $g \in G$ that can be unambiguously phased **do**
    let $h + h'$ explain $g$
    add $h$ and $h'$ to H
    remove g from G
**end for**
**while** $\exists$ h $\in$ H : $h + h' = g \in G$ for any $h'$ **do**
    let $h + h'$ explain $g$
    add $h'$ to $h$ if it is not already in h
    remove g from G
**end while**
**return** explanations

generate the set of all possible explanations, and can choose the "best" set of explanations. According to Halldorsson, there are three different metrics that we can use to decide if a set of explanations is a good set of explanations:

- Minimize the number of orphans
- Maximize Unique Resolutions. This means that, given your set of explanatory haplotypes, the largest number of genotypes has a single explanatory pair.
- Minimize Inference Distance. This means that we use as few applications of Clark's rule as possible.

This is a short-range technique precisely because as the sequences the algorithm is applied to gets longer, the probability that a sufficient number of genotypes can be resolved decreases. A randomized version of this algorithm would be to start from some permutation of the data, or subset of permutations, and choose from among the explanations generated by the subset. There are a variety of ways to choose the permutation to start with, perhaps by sorting the haplotypes in order from least number of ambiguous sites to most number of ambiguous sites, or by some other means.

### 2.2.2 ML Phasing

Another way we can choose a "good" explanation is to choose the most likely set of haplotypes. To define this problem well, we first assume a uniform distribution on all of the genotypes in G. This implies that the probability of some genotype $g \in G$ is $n\phi^g$, where $n$ is the number of times that genotype occurs in $G$. We find that

$$L = \prod_{i=1}^{n} \phi^{g_i} = \prod_{i=1}^{n} \sum_{h+\overline{h}=g_i} \phi_h \phi_{\overline{h}}$$

where $\phi_{h_i}$ is derived from the frequency of the haplotype $h_i$ in the set $H$. As in Clark phasing, this encourages the use of the same haplotype to explain multiple genotypes. Calculating this likelihood involves enumerating all possible haplotypes that explain each genotype and performing an EM algorithm on that set given an arbitrary set of initial frequencies on the haplotypes.

**Algorithm 2** EM Phasing

```
1: for unique genotype g ∈ G do
2:     for explanation (h, h') of g do
3:         add h and h' to H if it is not already in H
4:     end for
5: end for
6: Initialize haplotype frequencies randomly
7: while the likelihood has not converged do                ▷ Expectation
8:     for each genotype g do
9:         for each explanation (h, h') do
10:            Use φ_h, φ'_h to calculate φ_g
11:        end for
12:    end for                                              ▷ Maximization
13:    for each genotype g do
14:        for each explanation (h, h') do
15:            Use φ_g to find the most likely values of φ_h and φ'_h
16:        end for
17:    end for
18: end while
19: Use the most likely explanation to explain each genotype
20: return explanations
```

This yields exact solutions, but requires exponential time and memory since it must enumerate and iterate over all possible explanations of the genotypes. As in Clark phasing, this quickly becomes intractable.

## 2.3 Long Range Phasing

A Long-Range Phasing problem generally has a thousand or more different alleles in a particular genotype that we want to phase. For long-range phasing, we are considering site counts in the thousands. Heuristics are required to drastically reduce the search space of possible haplotypes from $2^m$ (where $m$ is the number of ambiguous sites) to a more tractable quantity. The fundamental insight required to construct a heuristic for long-range phasing is the same as in short range phasing - we want individuals to share haplotypes. The haplotypes in the long-range case, however, are too long for us to reasonably expect to find a whole genotype that can be unambiguously phased, or for us to be able to effectively enumerate all of the explanations for each genotype.

It is enough, then to relax our condition of "organisms should share haplotypes" to the condition, "organisms should share parts of haplotypes". This

assumption is well supported by biology, which makes the assumption that all human beings are descended from a common ancestor, and the theory of evolution provides a means for which different identical segments of DNA can move, change, and be exchanged in the population, through the mechanisms of mutation and recombination. We therefore turn our attention to methods of clustering haplotypes in order to reveal this sharing and highlight those substrings of DNA that may have been inherited from a common ancestor.

This is achieved by appealing to the coalescent model, which models relationships between human beings as having been constructed via a tree structure where a single shared ancestor of a population is the source of a particular mutation that is present in a large proportion of the population, and as a result of this inheritance haplotypes descended from the same parent share the same mutations. Recombination events result in entire segments of DNA 'staying together' when they are passed on and independently segregate, and that genotypes of individuals that are not related to each other at the current generation may be able to inform each other's haplotype phase thanks to IBD (identity by descent). In particular, these methods take advantage of segments of DNA which are in linkage disequilibrium - that is to say, two different alleles $i$ and $j$ will appear together with a probability that is different from the product of the probability of $i$ times the probability of $j$.

$$D_{AB} = p_{AB} - p_A p_B$$

A group of SNPs in linkage disequlibrium with one another forms a haplotype block, a region of DNA that contains a sequence that is very likely to appear in another human being's genome sequence. This results in individuals that have the same segment of a haplotype as one another. By identifying the limited sample space of these substrings, we are able to drastically reduce the number of possibilities that we will consider as potential solutions to the phasing problem.

## 2.4 Software Packages for Long-Range Phasing

An excellent review of different algorithms that have been implemented to perform long-range phasing using these coalescent-based methods can be found at [Browning and Browning, 2011], but for completeness' sake I shall repeat them here.

A package called Arlequin implements the EM algorithm, a package called HAPLOTYPER[Niu et al., 2002] implemented Gibbs sampling to maximize a partition-ligation strategy, and a package called PL-EM implements Partition-Ligation Expectation Maximization. Partition-Ligation is a method of extending the number of ambiguous sites that EM can phase by partitioning a long string into short strings that are each individually phased (partitioning) then putting them back together (ligating). Population-based statistical phasing methods, or, methods that attempt to infer the coalescent tree that gave rise to the haplotypes, are implemented by PHASE [Stephens and Scheet, 2005],

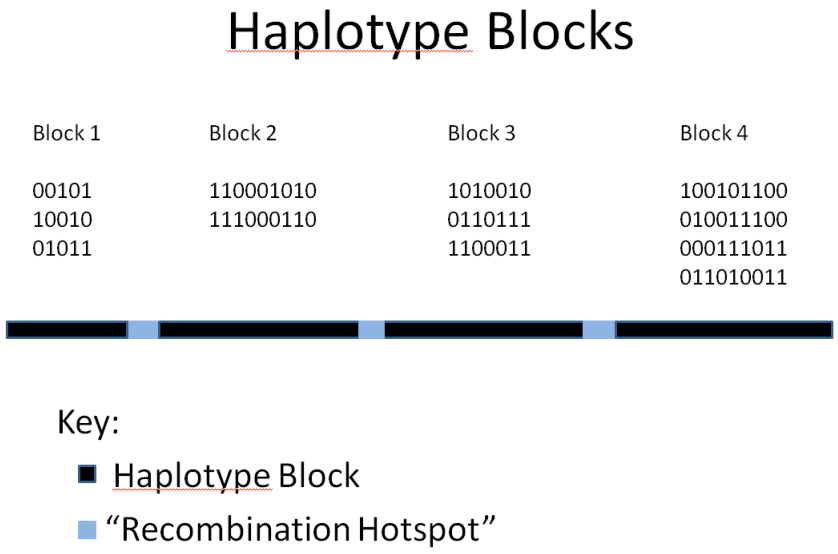



Figure 1: Often there will only be a small number of haplotype sequences that arise in a "block" of DNA, separated by areas where recombination is common. We call such segments haplotype blocks.

fastPHASE [Scheet and Stephens, 2006], MACH [Li and Stephens, 2003], and IMPUTE2 [Howie et al., 2009].

BEAGLE [Browning and Browning, 2007b] is a software package that, rather than explicitly modelling the recombination and mutation that happens in the coalescent tree, attempt to capture haplotype blocks through a method of haplotype clustering that relies on what is called an Acyclic Probabilistic Finite Automata. HapROSE contains an implementation of the inference algorithm used by BEAGLE.

Most recently, an algorithm called EAGLE [Loh et al., 2015] was able to perform quick and accurate phasing on a large cohort in the UK, a population with a large and diverse set of haplotypes. It accomplished this by using IBD information and by performing inference on an HMM with "aggressively pruned" hidden states to keep the problem tractable.

# 3 Haplotype Clustering for Long Range Phasing

## 3.1 Inferring Haplotypes from Haplotype Clusters

There is a long history of using the concept of haplotype clusters to aid in solving the phasing problem. This approach has its roots in the Li and Stephens framework, which uses the concept of clusters to create an HMM whereby each genotype is emitted by some hidden haplotype cluster. Their key insight was that regions of linkage disequilibrium among multiple SNPS was indicative of the recombination process, and that by considering all SNPS that are in LD in a local region we can be used to infer SNPS in an unrelated individual in the population. This idea of using "surrogate parents" to aid in haplotype phasing was elucidated and built upon by the algorithms to come, but the general core of the theory is that haplotypes can be thought of as a set of loosely linked alleles that enter different clusters, or blocks, and switch between them at intervals.

## 3.2 PFA

One way to model haplotype clustering is to use a data structure known as the Acyclic Probabilistic Finite Automata [Ron et al., 1995]. To understand these, we must first wrap our heads around the more general PFA (Probabilistic Finite Automata).

PFA's are finite state machines which generate strings in a probabilistic manner. Since, to capture the behavior of DNA, we are interested only in finite strings, we will consider a PFA that has both a final symbol and a final state. Beginning at some start state, we transition to some other state provided by the transition function, with a probability determined by the transition probability function of the PFA. In this manner, we traverse the PFA until we hit an end state, in which case the string terminates.

More formally, a PFA $M$ is a 7-tuple $(Q, q_0, q_f, \Sigma, \zeta, \tau, \gamma)$ where

- $Q$ is a finite set of states

- $q_0 \in Q$ is the starting state
- $q_f \in Q$ is the end state
- $\Sigma$ is a finite alphabet
- $\zeta \notin \Sigma$ is the final symbol
- $\tau : Q \times \{\Sigma \cup \{\zeta\}\} \to Q \cup \{q_f\}$ is the transition function
- $\gamma : Q \times \{\Sigma \cup \{\zeta\}\} \to [0, 1]$ is the next symbol probability function

such that $\forall q \in Q \sum_{\sigma \in \Sigma \cup \{\zeta\}} \gamma(q, \sigma) = 1$, that is, all of the probabilities of transitioning from some state $q$ to any other state are probabilities that sum to 1. We must be able to reach the final state from any other state $q$ with nonzero probability.

We generate a finite string on the alphabet $\Sigma$ by starting from starting state $q_0$, then choosing the next state $s_{i+1}$ in the string using the next state probability function function $\gamma(q_i, s_{i+1})$. We then add to the string the symbol generated by $\tau(q_i, \sigma)$ Therefore, the probability of any particular string is

$$P^M(s) = \prod_{i=0}^{l-1} \gamma(q_i, s_{i+1})$$

.

## 3.3 APFA

An APFA (Acyclic Finite Probabilitic Automata) is a variant of PFA. As the name implies, it does not contain any cycles. Additionally, we will restrict our analysis to levelled APFA - all nodes in a levelled APFA are associated with a certain depth. The total depth of such an APFA is defined as the length of the longest path between the start state $q_0$ and the end state $q_f$. We define the start state as being at depth 0. and the final state as being at depth $m$. Such an APFA generates strings of length up to $m$. A key claim of the phasing algorithm implemented by BEAGLE and hapROSE is that this structure is a convenient way to capture LD between SNPs, and can be used to great effect in order to infer the phase of a genotype. If some SNP $s_1$ is in strong LD with a sequence of other SNPs immediately after it in the string, then we can learn an APFA that generates a string where the symbol associated with $s_1$ transitions through the sequence of other observed alleles with probability proportional to its observed frequency in the sample. This framework possesses inherently markovian properties that we will leverage in the inference algorithm. Phrased in a different way, we assume that there is some APFA that generates the haplotypes which are conflated to become the observed genotypes we are attempting to phase, and our goal is to try to learn that APFA.

## 3.4 Learning an APFA

Let us consider a set of inputs, $n$ strings of length $m$, that are generated by an APFA. We are interested in inferring the generating APFA from the input strings. We first establish a measure of how "far away" two APFAs are from one another - Kullback-Liebler divergence,

$$D_{KL}[P^M||P^{\hat{M}}] = \sum_{s \in \Sigma * \zeta} P^M(s) log \frac{P^M(s)}{P^{\hat{M}}}]$$

It is important to note that while not a true distance metric (it does not obey the triangle inequality), if the KL divergence between two APFA is 0 then the APFA are equivalent, which means that if we were given an infinite number of strings from the first APFA and an infinite number of strings from the second APFA we would not be able to tell that there were two different generating APFA. So, our algorithm does a good job if it can produce an inferred APFA with a $KL$ divergence of less than some $\epsilon$ away from the target APFA.

## 3.5 BEAGLE and Ron et. al. Clustering Algorithms

Ron et. al. [Ron et al., 1995] proposes an algorithm for learning an APFA and rigorously proves that the algorithm outputs a good hypothesis with high probability, and that this can be accomplished efficiently in time polynomial in its parameters. Specifically, we define the following:

- $\delta$, a confidence parameter
- $\epsilon$, an accuracy parameter
- $\mu$, a distinguishability parameter
- $n$, an upper bound on the number of states in the APFA
- $\Sigma$, an alphabet

Ron's learning algorithm outputs with probability $1-\delta$ an $\epsilon$-good hypothesis with respect to the target APFA in time polynomial in $1/\epsilon$, $\log 1/\delta$, $|\Sigma|$, $n$, and $1/\mu$.

Both the BEAGLE and Ron implementations use the same basic framework. We begin by constructing a tree from the input strings (haplotypes in our particular application). This is done as follows:

Simply stated, the sample tree is a tree where every haplotype is represented as a chain of nodes from root to leaf, and the edge weights between nodes correspond to the number of times a haplotype appears.

After this tree is constructed, we look for any two nodes in the tree such that the subtrees rooted at those two nodes generate strings with a probability that is indistinguishable from one another. We then say that those two nodes are similar, and "collapse" the tree by folding (joining) the two subtrees into a single subtree that has the combined counts of both.

**Algorithm 3** Build Sample Tree

```
for h ∈ H do
    Begin at the root
    while h has a next character c do
        if !currNode.hasChild(c) then
            currNode.addChild(c)
        end if
        edgeCount(currNode, currNode.getChild(c)) += 1
        currNode ← currNode.getChild(c)
    end while
end for
```

**Algorithm 4** Infer APFA

```
Let D be the maximum APFA depth
Let S be the set of input strings
Construct Sample Tree from S //initial state of the APFA we are to learn
Initialize d(i) to 0 //current depth in APFA
while d(i) < D do
    Find nodes j and j' from level d(i) such that
    • A significant number of sample strings pass through both nodes
    • The nodes are Similar, as determined by a call to SIMILAR(j,j')
    if A similar pair of nodes is found then
        FOLD(j,j')
    else
        d(i) = d(i+1) //inspect the nodes at the next depth
    end if
end while
return APFA
```

These similarity functions will be our primary focus, but before we can dive more deeply into how they behave we must first more formally define some quantities.

- Let $G$ be the graph at its current state of collapse
- Let $j$ be the current node
- Let the edge exiting $j$ that corresponds to emitting symbol $\sigma$ be denoted as $m_j(\sigma)$, $\forall \sigma \in \Sigma \cup \zeta$
- Associate with each such edge a count $m_j(\sigma) \in \mathbb{N}$ that counts how many strings in the input set pass through that edge.
- If edge $m\sigma$ is a directed edge from node $j$ to some other node $j'$, then let the transition function $\tau(j, \sigma) = j'$

We fold two nodes $i$ and $j$ together by pointing all of both node's parents to the same node, combining their outgoing edge counts, and recursively folding their children. This can be interpreted as meaning that, regardless of whatever path through the APFA was taken to get to either node $i$ or node $j$, once you are at either of those nodes the probability that the string will have a certain suffix is equal, or at least, is very similar.

**Algorithm 5** Fold Nodes

```
for all nodes with edges that go to j' do
    set the corresponding edge to end at j
end for
for all symbols σ ∈ Σ ∪ ζ do

    if then m_j(σ) = 0 and m_j'(σ) > 0
        Set e to start at j
    end if
    if then m_j(σ) > 0 and m_j'(σ) > 0
        FOLD(τ(j, σ), τ(j, σ))
    end if
    m_j(σ) ← m_j(σ) + m_j'(σ)
    Remove j' from G
end for
```

## 3.6 Ron Similarity Function

Ron's similarity function is as follows:

The main idea of this algorithm is that it calls two nodes $i$ and $j$ similar if and only if they are $\mu$ indistinguishable, which means that the probability of any suffix that could arise after node $i$ is indistinguishable from the probability that of any suffix after node $j$.

**Algorithm 6** Ron Similarity

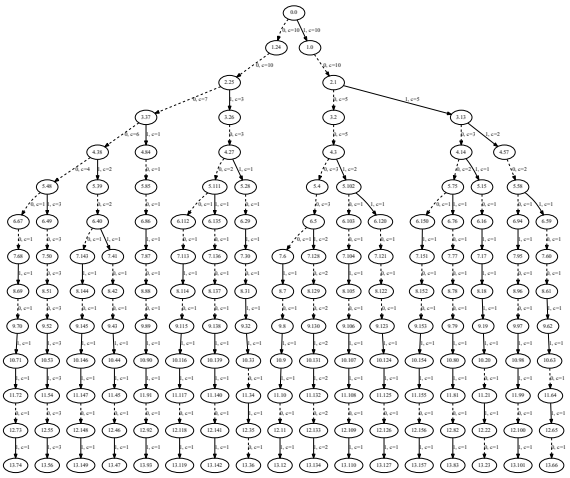
**Require:** $u, p_u, v, p_v$
**if** $|p_u - p_v| \geq \mu/2$ **then**
  **return** non-similar
**else**
  **if** $p_u < \mu/2$ and $p_v < \mu/2$ **then**
    Return similar
  **else**
    **for** $\sigma \in \Sigma \cup \zeta$ **do** ▷ Calculate probabilities
      $p'_u = p_u m_u(\sigma)/m_u$
      $p'_v = p_v m_v(\sigma)/m_v$
      $u' \leftarrow \tau(u, \sigma)$
      $v' \leftarrow \tau(v, \sigma)$
      **if** SIMILAR(u', p'u, v', v'u) = non-similar **then**
        **return** non-similar
      **end if**
    **end for**
  **end if**
**end if**
**return** similar

## 3.7 BEAGLE Similarity Function

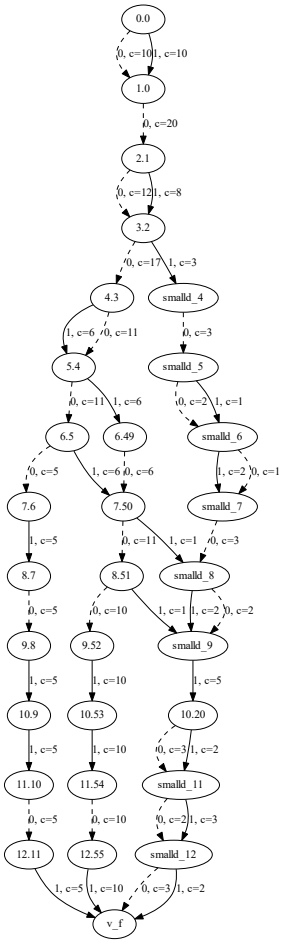
A key difference between Ron's similarity algorithm and BEAGLE's similarity algorithm is that, where the Ron algorithm returns a true or false similarity judgement, the BEAGLE algorithm assigns to each pair of nodes a real-valued similarity score. This score is calculated as a function of the edge counts entering and leaving the node. In particular, it looks at all corresponding subtrees of two nodes, and determines the maximum similarity score between any of the subtrees. If this number is below a certain threshhold, which is also a function of the total number of incoming edges of the two nodes being compared, then the nodes are said to be similar. BEAGLE merges all pairs of nodes at a particular depth of the APFA from 0 to $D$, in order from highest to lowest similarity score.

## 3.8 Clustering Accuracy: Ron vs BEAGLE

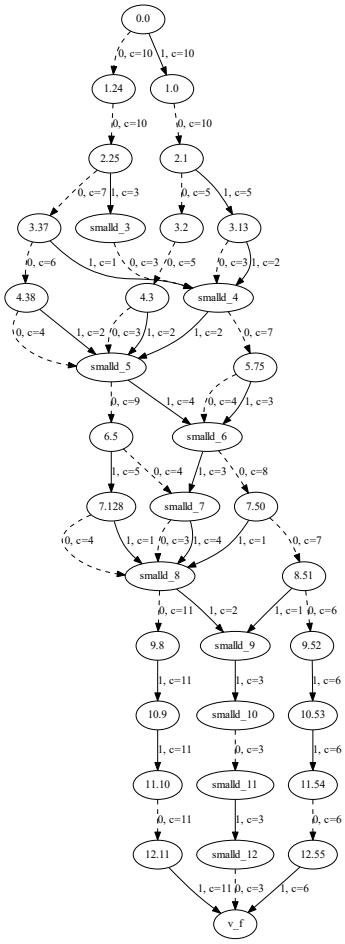
The primary difference in these two heuristics is how they perform on reference panels with fewer samples. As the number of training haplotypes increases, we expect both algorithms to learn APFA that become increasingly similar to one another. The BEAGLE algorithm consistently produces APFA with fewer edges than the Ron algorithm for lower numbers, while they both converge to the same APFA at higher numbers of training samples. The APFA depicted in Figure 4 is the "correct" APFA, and we see that the Ron algorithm with a $\mu$ parameter of 0.5 has a difficult time approximating it at lower sample counts.



(a) Sample Tree for 10 training haplotypes

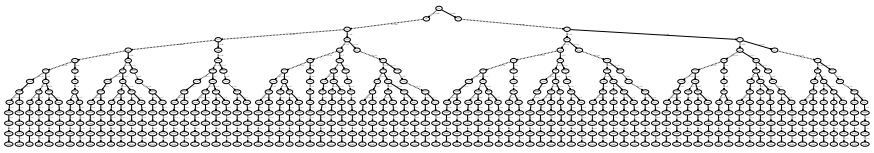



(b) BEAGLE APFA

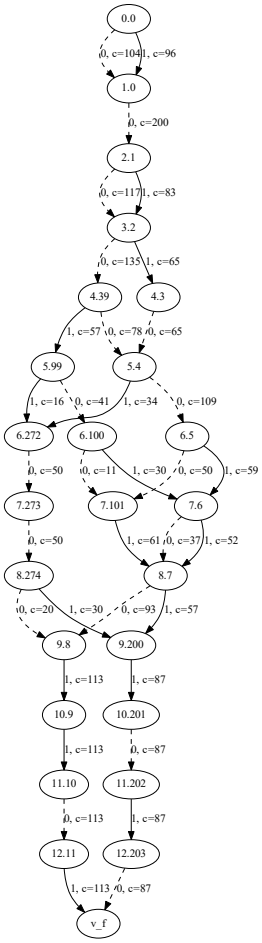



(c) Ron APFA

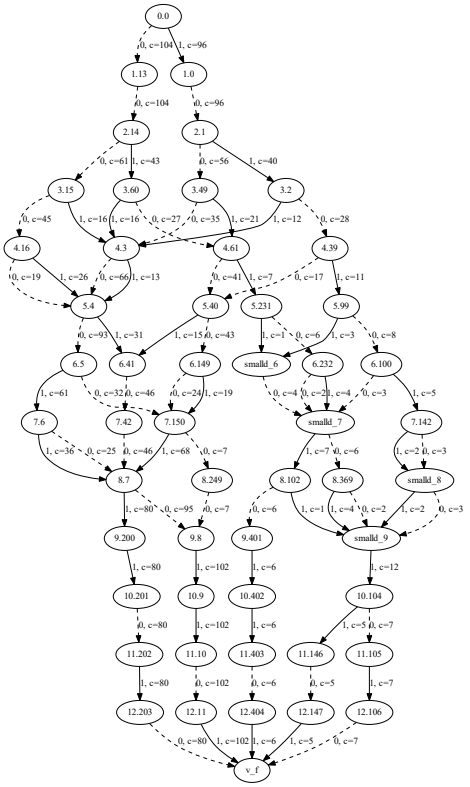
Figure 2: Ten samples, short genome (15 alleles)



(a) Sample Tree for 100 training haplotypes

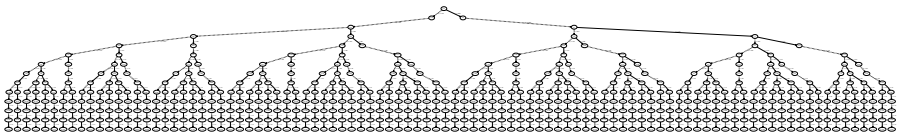



(b) BEAGLE APFA

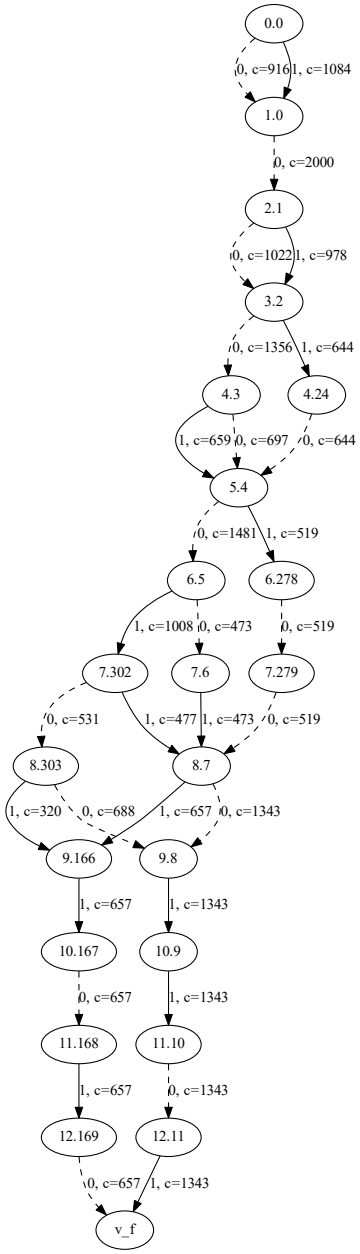



(c) Ron APFA

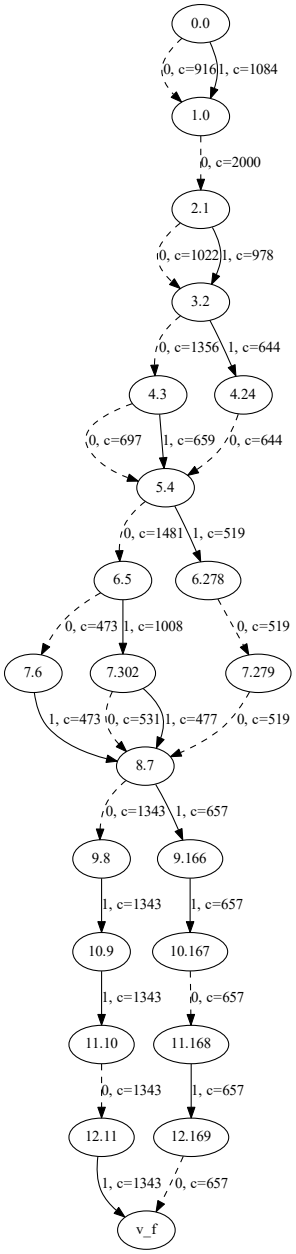
Figure 3: Hundred samples, short genome (15 alleles)



(a) Sample Tree for 1000 training haplotypes

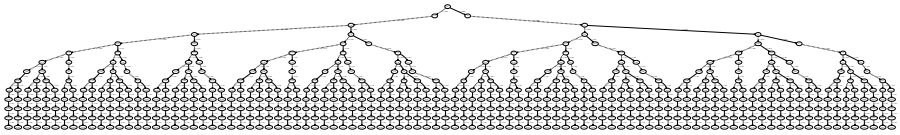



(b) BEAGLE APFA

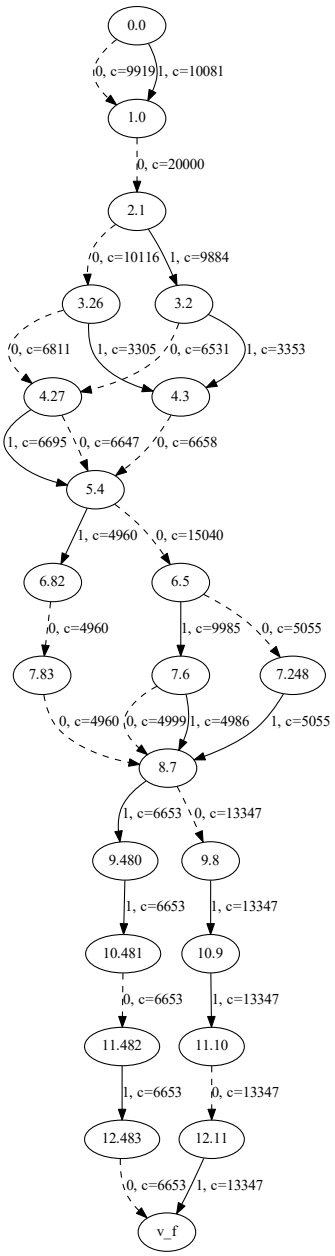



(c) Ron APFA

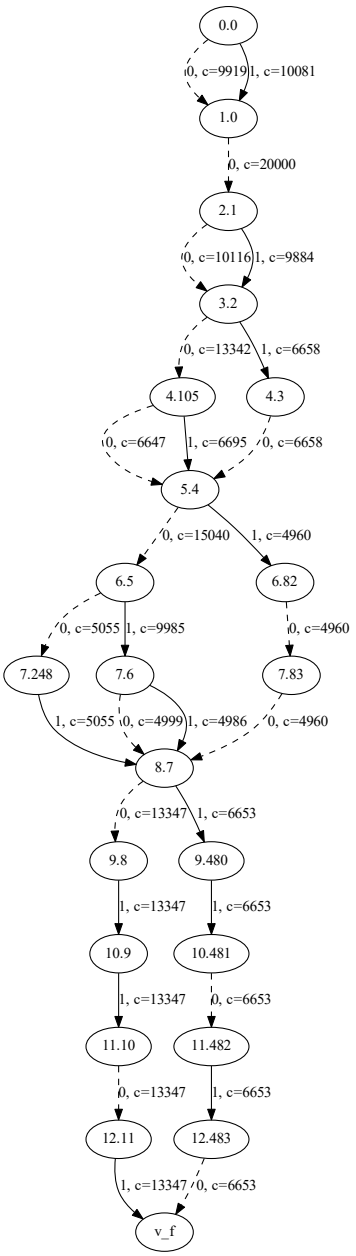
Figure 4: Thousand samples, short genome (15 alleles)



(a) Sample Tree for 10000 training haplotypes




(b) BEAGLE APFA




(c) Ron APFA

Figure 5: Ten Thousand samples, short genome (15 alleles)

**Algorithm 7** BEAGLE Similarity

**Require:** Node $u$, Node $v$, Double $n_x$, Double $n_y$, Double threshhold

$u\_totalcount \leftarrow \sum_{\sigma \in \Sigma \cup \zeta} m_u \sigma$
$v\_totalcount \leftarrow \sum_{\sigma \in \Sigma \cup \zeta} m_v \sigma$
**if** $u\_totalcount = v\_totalcount = 0$ **then**
    Return 0
**end if**
**if** $u\_totalcount = 0$ **then**
    **return** Max difference between edges of v
**end if**
**if** $v\_totalcount = 0$ **then**
    **return** Max difference between edges of u
**end if** ▷ Both u and v have outgoing edges with nonzero counts
localdiff(u,v) $\leftarrow \max_{\sigma \in \Sigma \cup \zeta}$ abs$(m_u(\sigma)/n_x)$, abs$(m_v(\sigma)/n_y)$)

# 4 Phasing using a Learned APFA

To phase genotypes using learned APFA, hapROSE implements a multithreaded modified stochastic EM algorithm of the same kind proposed by Browning and Browning [Browning and Browning, 2007b] on an HMM that is built using the edges of the learned APFA. Observed genotypes are treated as a series of observations, and seek to find the underlying state that is most likely to have generated those observations. Each state in the HMM is a pair of edges, since we are seeking the most likely pair of alleles that generated a particular allele in the genotype at a given point in the sequence.

The algorithm samples potential generating haplotypes for a given genotype with a probability proportional to the posterior distribution of each allele at each position. This is inferred using the fwd-bwd algorithm on the underlying inhomogenous diploid HMM. These sampled haplotypes are then used to train another APFA in the next iteration of the algorithm. This process is repeated for a number of iterations - hapROSE adopts the number that BEAGLE recommends, which is 10. After the final iteration of sampling from the forward-backward algorithm, the most likely pair of haplotypes is chosen using the viterbi algorithm on the final APFA.

Smoothing transition probabilities and pruning edges of the APFA has a drastic effect on the speed of the algorithm - since we must consider all possible pairs of edges, the runtime quickly becomes intractable on even short strings if the input strings are very diverse. Let $k$ be the largest number of edges at any given depth in the APFA - this algorithm's running time is $O(\binom{k}{2}^2 m$, where $m$ is the number of alleles in a genotype. This first binomial term squared quickly dominates the runtime. BEAGLE's variant is faster precisely because it relaxes collapse conditions for lower node counts, when we are unlikely to have observed a sufficient number of strings for two nodes to be $\mu$-indistinguishable, in the Ron framework. Methods to prune this tree while preserving those paths most likely

to have generated haplotypes are an excellent direction for future study.

## 4.1 The Diploid HMM

We now turn our attention to formally defining the inhomogenous diploid hmm, and the phasing algorithm being used to infer genotype phasings.

An HMM [Rabiner, 1989] is a 5-tuple $(O, X, \tau, \gamma, \pi)$ where

- $X$, the set of hidden states in the model
- $Y$, the set of emitted symbols in the model
- $\tau$, the state transition probability function
- $\gamma$ is the observation emission probability function
- $\pi$ is the initial probability function

## 4.2 The Phasing Algorithm

**Algorithm 8** Stochastic EM

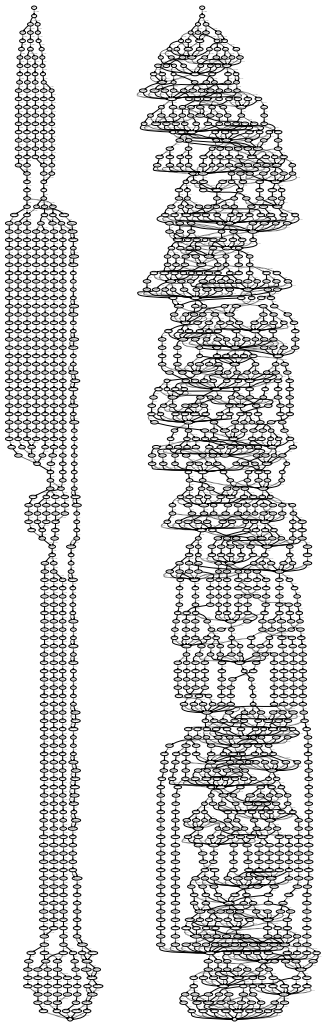
```
1: Construct Sample Tree
2: Learn APFA
3: Construct HMM
4: for i iters do
5:     sampledHaplotypes ← fwd_bwd(genotypes)
6:     sampleTree ← sampledHaplotypes
7:     Learn APFA
8:     Construct HMM
9: end for
10: sampledHaplotypes ← viterbi(genotypes)
11: return sampledHaplotypes
```

The phasing algorithm treats the genotype as a series of observations $O_1, O_2, \ldots :$ $O_i \in Y \forall i$ emitted by an HMM, whose underlying states $X$ are pairs of edges in the APFA at a particular depth. The transition probabilities from one state to another depend on whether the parent nodes in one state are the child nodes of the previous state, and if so, the probability $\tau$ is computed as a function of the edge counts. The initial probability function $\pi$ is uniform.

Phasing proceeds by running the forward-backward algorithm on this HMM, then sampling a potential explanatory pair of haplotypes from the posterior. This process is repeated for a number of iterations - BEAGLE recommends 10. Afterwards, the viterbi algorithm is run in order to get a maximum likelihood estimate.

The forward-backward step significantly increases the size of the inferred APFA, since it samples directly from the posterior at a given position, and does

not take into account whether a state could have been transitioned to from the previous state. Figure 5 shows how drastically an APFA can change after a single round of resampling. This increases the runtime in future iterations, but it reduces the likelihood of the model overfitting the reference haplotypes.



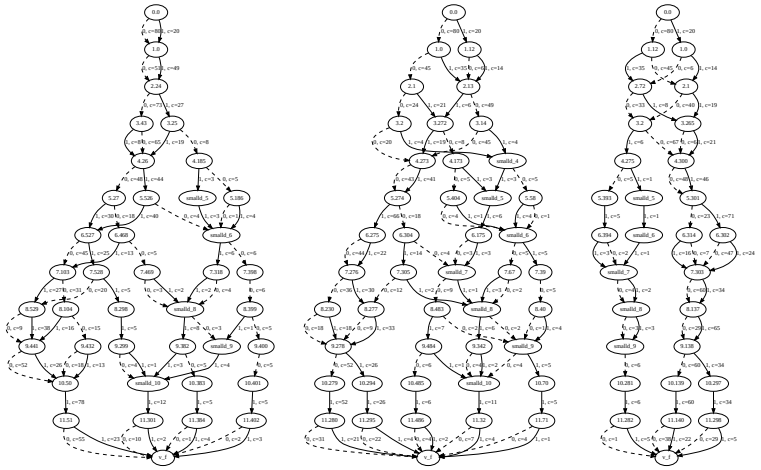
(a) Before Resampling (b) After Resampling

Figure 6: Resampling Complicates the Learned Apfa

## 4.3 Iterative Sampling

In order to gauge the performance of the different heuristics across the different iterations of the phasing algorithm, I took as my set of haplotypes the sequences given in Browning and Browning's 2006 paper, in particular, the idealized data that they were using for figure 1. I randomly created genotypes out of those haplotypes, and allowed hapROSE to use the true haplotypes as a reference panel to start the first iteration of the algorithm.

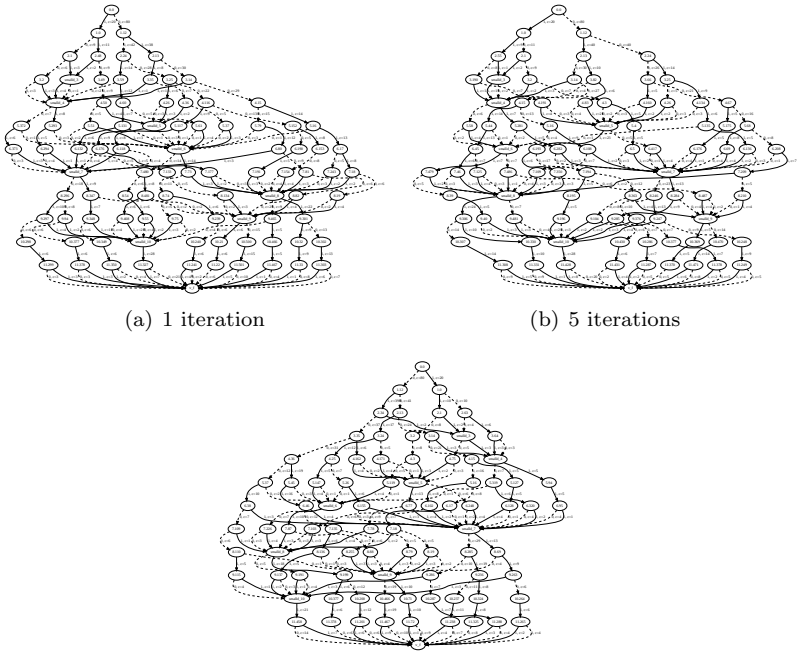
The Ron algorithm was run with a $\mu$ indistinguishability parameter of 0.5, indicating that it would only perform a node collapse if the probability distributions downstream of a node were very similar. The resulting APFA reflect this, as the Ron APFA consistently contains more states at every level in every iteration.



(a) 1 iteration (b) 5 iterations (c) 9 iterations

Figure 7: Node Collapse after Repeated Resampling (BEAGLE Heuristic)

:



(a) 1 iteration

(b) 5 iterations

(c) 9 iterations

Figure 8: Node Collapse after Repeated Resampling (Ron Heuristic)

:

## 4.4 Phasing Accuracy: Ron vs BEAGLE

The next thing that needs to be ascertained is how well the phasing algorithm performs when it is performing inference on an APFA that is being learned using the BEAGLE variant, compared to how accurate phasing is when it is being performed on an APFA that is being learned using the Ron variant. For this section, all tests were performed on a machine running an Intel(R) Core(TM) i7-4700MQ CPU @ 2.40 GHz (8 CPUs), with 8192MB of RAM.

Let us define the **switch error** as the fraction of positions for which the phase between the two haplotypes is different relative to the previous position. [Bansal et al., 2008] [Lin et al., 2002]. In other words, it is the number of times in an inferred phasing that the alleles in the true phasing are wrongly assigned, and we have to switch over to the other haplotype. In this section I explore the switch error of the two approaches on the same input training data, as estimated using a monte carlo approach. I do this for both short-range(100 alleles) and long-range (1000 alleles) phasing problems. The haplotypes that I was sampling from were generated by the hudson ms simulator. [Hudson, 2004]

For many of the following trials, I was using a reference panel of 44 haplotypes to phase 22 randomly generated genotypes. These numbers were chosen

due to memory constraints on the machine used to benchmark performance.

The distinguishability parameter $\mu$ that controls the threshhold at which the Ron et al. algorithm collapses nodes is chosen in all cases to be what I empirically felt would be a good balance between speed and accuracy, and performs at a speed comparable to that of BEAGLE.

### 4.4.1 Short-Range Phasing

The pool of sampled haplotypes was generated using the hudson simulator by running the command

```
ms 100 1 -t 5 -s 100 -r 100.0 2501
```

Randomized genotypes were then created from randomly selected haplotypes. These haplotypes became the reference panel to start the iterations, and the genotypes were then phased using the stochastic EM algorithm.

To compare the effect of phasing 22 genotypes to the effect of phasing 50 genotypes, I needed to relax the value of $\mu$ to 0.75 so that the Ron algorithm would terminate in a timely manner.

Table 1: EM results - 22 genotypes of length 100

| Heuristic | Num Phased | Time(mins) | Switch Error | Var |
|---|---|---|---|---|
| Ron, $\mu = 0.5$ | 22 | 2 | 0.056 | 4.9E-4 |
| Browning | 22 | 1.5 | 0.067 | 5.3E-4 |
| Ron, $\mu = 0.75$ | 50 | 57 | .058 | .001 |
| Browning | 50 | 52 | 0.046 | 4E-4 |

### 4.4.2 Long-Range Phasing

As in the short range phasing case, I simulated haplotypes using the Hudson ms simulator, as follows:

```
ms 100 1 -t 5 -s 1000 -r 100.0 2501 > 100samps_len1000.txt
```

Scaling up to long range phasing led to a slowdown in the Ron algorithm as well. I ran it at both $\mu = 0.5$ and $\mu = 0.75$ in order to compare the results.

Table 2: EM results - 22 genotypes of length 1000

| Heuristic | Num Phased | Time(mins) | Switch Error | Var |
|---|---|---|---|---|
| Ron, $\mu = 0.5$ | 22 | 43 | .044 | -7E-8 |
| Browning | 22 | 37 | 0.049 | 1.8E-5 |
| Ron, $\mu = 0.75$ | 22 | 37 | .039 | 9E-5 |
| Browning | 22 | 30 | 0.03 | 8E-5 |

We see that at the tighter threshhold, the Ron algorithm takes a bit longer to perform the inference, but has a lower switch error. At the looser threshhold, it still takes longer to perform the inference, but now it has a higher switch error.

## 4.5 Scaling with Number of Reference Haplotypes

To compare how the Ron heuristic would compare to BEAGLE when run at a comparable speed on different sized reference panels, I relaxed the $\mu$ parameter in order to encourage more node collapse in the Ron algorithm, speeding the algorithm up dramatically. I also ran it on a shorter set of genotypes to enable me to perform thousands of simulations without the task becoming prohibitively time consuming.

What follows are the results of performing 100 separate phasings of 10 iterations each on both a small number of reference haplotypes, and a large number of reference haplotypes.

### 4.5.1 Small number of reference haplotypes

To generate this data, I performed phase inference on the same 50 randomly sampled genotypes for both Ron and BEAGLE, and calculated the mean switch error and variance. I did this for a short-range dataset of 15 alleles, designed to have 3 haplotype blocks.

Note that in this case, unlike in previous tables, the switch error is averaged across 100 independent runs of the algorithm, and the variance refers to the variance of the average switch error.

Table 3: EM results - 50 genotypes of length 15, repeated 100 times

| Heuristic | Num Phased | Time(mins) | Avg(Switch Error) | Var |
|---|---|---|---|---|
| Ron, $\mu = 1.15$ | 50 | 5 | .125 | .009 |
| Browning | 50 | 4 | 0.097 | 0.007 |

### 4.5.2 Large number of reference haplotypes

The next question was to test how the algorithms performed 100 separate phasings of 10 iterations each on the same 200 randomly sampled genotypes for both Ron and BEAGLE, and calculated the mean switch error and variance. I did this for a short-range dataset of 15 alleles, designed to have 3 haplotype blocks.

Table 4: EM results - 200 genotypes of length 15, repeated 100 times

| Heuristic | Num Phased | Time(mins) | Avg(Switch Error) | Var |
|---|---|---|---|---|
| Ron, $\mu = 1.07$ | 200 | 12:40 | .130 | .010 |
| BEAGLE | 200 | 14:30 | 0.079 | 0.005 |

### 4.6 Summary of Results

For short and long range phasing, the Browning heuristic outperforms the Ron heuristic at the chosen parameter in almost every case in terms of speed, but is less accurate in terms of accuracy.

When tests were ran to determine how they scale to the size of the reference panel, the Ron algorithm slowed down dramatically, so I was forced to repeatedly adjust the $\mu$ threshhold in order to encourage a higher speed. Predictably, the Ron et. al. algorithm ran faster with a larger distinguishability parameter, but was much less accurate and exhibited a higher variance. Tuning this parameter to get a good balance of phasing and speed is a difficult task, and on larger reference panels BEAGLE's dynamic adjustment of the similarity threshhold is welcome. On smaller reference panels on the order of two dozen individuals, Ron's algorithm runs in comparable speed and with higher accuracy.

## 5 Tracts for Genotype Phasing and Inference

An important tool that we can leverage to help us phase long range is the concept of a shared tract between individuals. A tract is a substring shared by multiple haplotypes in a set, with the additional property that it must begin at the same position $i$ and end at the same position $j$ in each haplotype. A tract is said to be a maximal shared tract if it cannot be extended in either direction.

This tract information can be used to great effect when inferring haplotype phase. It is possible to generalize the concept of Clark Phasing to that of a Clark Consistency Graph [Halldorsson et al., 2011], a structure that models the relationship between some haplotypes that may share a sequence IBD, and allows us to phase a large number of genotypes if a sufficient number of shared haplotypes between them are known.

This is done by drawing an edge between any two genotypes that are known to share a tract - any cliques in this graph indicate that the tracts can be used to reliably "vote" on the most likely phasing of a particular allelic substring in an individual. This implies a method for phasing that bypasses the need to construct an APFA solely using a similarity function, instead relying on known shared tracts in order to determine its structure.

There is currently a wide variety of methods for finding tracts available to us, and they are discussed here briefly. Yet, many of these methods require a reference panel of already phased haplotypes, in a similar way that many long range phasing algorithms require some sort of reference panel in order to begin to

cluster haplotypes. Detecting tracts in genotypes is a challenging computational problem because genotypes contain ambiguous characters, so you run into the same potential pitfalls as in phasing - if you encounter an ambiguous site, the number of potential tracts present can skyrocket. When two genotypes match one another, there is no guarantee that this is transitive - a third genotype can match both of those genotypes, and yet all three together may not share the same haplotype. As an example, consider the genotypes

```
0012021
0210011
0210021
```

They match one another, in that of the four haplotypes that compose those two genotypes, there is potentially one haplotype from one of the genotypes can share a tract with one haplotype from the other genotype. For example, consider the phasing

```
0011001
0010011

0010011
0110011

0110011 OR 0110001
0010001    0010011
```

We notice that the second haplotype of the first genotype is identical to the first haplotype of the second genotype. Likewise, we can choose some phasing of the third genotype that will result in a matching tract with either the first genotype or the second genotype. We cannot, however, identify a tract that is common to all three.

When presented only with genotypes this sort of rough approximation that identifies a potential tract is a good place to begin. We may also be able to ignore the wildcard characters and only consider those homozygous portions of genotypes where multiple haplotypes agree - in this case, a shared tract is unambiguous. Since we do not know the phasing of genotypes a priori, a single genotype can match other genotypes, in fact a very large number of other genotypes, very easily. This is due to the fact that they can have any number of a large number of potential generating haplotypes. As a matter of fact, as was proven earlier in this paper, a single genotype can have up to $2^n$ generating haplotypes, where $n$ is the number of different ambiguous sites that are present in the genotype. Yet, finding tracts is so important because we can use them to help us infer phase because we are using a powerful coalescent model which allows us to map different people who may be homozygous at some particular subsection of their genome to other people who are heterozygous, but may contain a haplotype in common with the individual that was homozygous. The underlying principle of the above inference algorithms is that the same

haplotypes, or subsequences of haplotypes, occur often in the population as a result of recombination.

Is there a way, then, to identify all such potential tracts in a set of genotypes? Current methods for finding tracts for haplotypes offer a good place to start.

## 5.1 Tractatus and Lumbertracts

Istrail Lab currently has an implementation of an algorithm called Tractatus, which is a suffix tree algorithm designed to find all tracts in a set of phased haplotypes in linear time. A colleague of mine, Daniel Seidman, is working on a way to generalize this result to haplotypes with wildcards, implying a potential mechanism for finding all possible tracts in genotypes.

## 5.2 PBWT and a Potential Variant

Recently, an algorithm called the Positional Burrows Wheeler Transform has been described, and it has been used to great effect in order to find all tracts in a set of haplotypes in linear time. [Durbin, 2014]

It does this by taking the set of haplotypes and going through all rotations of the resulting array, sorting the reverse prefixes of the haplotypes at each step and using the fact that after being sorted adjacent haplotypes are very similar to one another in order to reduce runtime while still finding set-maximal exact matches.

I am exploring a variation of this algorithm that promises to be able to identify potential tracts between genotypes - it involves a simple modification in the algorithm whereby, in sort order, the alphabet is ordered $0 < 2 < 1$. In the matching phase of the algorithm, we consider a 2 to match either a 0 or a 1. The rest of the algorithm proceeds as normal, except instead of simply comparing a string to a string above it (a downward pass) we must also compare strings to the strings below it (a second upward pass). This allows us to identify any places where a wildcard matches a 0 in the downward pass, and any places where a wildcard matches a 1 in the upward pass. This provides a mechanism for finding at least some matching tracts in genotypes, and I conjecture that it may be able to find all such potential shared tracts.

An issue with this method is that, as mentioned above, finding tracts is not pairwise transitive. More precisely, this method can find genotype substrings that are part of the same connected component in the Clark Consistency Graph, but does not identify cliques. We make this statement more precise as follows:

The different rotations $i = 0, \cdots, n-1$ of the PBWT allow us to extract information about the underlying Clark Consistency Graph on the subset of genome sequences that end at index $i$.

Consider a particular rotation of the PBWT and the accompanying sorted order of genotypes - we can partition the ordered genotypes into "match blocks", where the first genotype in the block contains the longest maximal match with an adjacent genotype in the block, and the final genotype in the block contains

the shortest nontrivial (a greater length than would likely occur by chance) maximal match with an adjacent genotype in the block.

Define a Clark Consistency Graph consisting of nodes that are sequences which begin where the shortest nontrivial maximal match begins, and ends at the position currently being considered by the rotation, with an edge between nodes if the sequences can share a haplotype. Since, the way the algorithm is defined, two adjacent genotypes can only have a match if they can share a haplotype, and since there is a unique length for the shortest maximal match in the set, then all adjacent genotypes in the block can share a haplotype, which is to say we can find a path in the graph from the first sequence to the last sequence.

We now turn our attention to the problem of finding any cliques in the graph, since the genotypes that comprise the vertices in the clique can be phased by positing that they each share a common haplotype, then taking the complement to complete the phase.

We begin by examining each match block, which we now know to be comprised of genotypes which are connected in the underlying graph that considers all sequences within the block up to the length of the shortest nontrivial match. If two non-adjacent genotypes in a match block can share a haplotype, then they share that haplotype with all genotypes in between them. The proof is by contradiction - let us assume two non adjacent genotypes share a haplotype with each other but not with every genotype in between. One of the two genotypes must come first in the sort ordering, and it must share a haplotype with the next genotype in the match block, and that one must share a haplotype with the adjacent one, and so on and so forth until the second genotype, or they wouldn't be part of the same match block. Two genotypes do not match when there is a homozygous position in one genotype, and the opposite allele is homozygous in the other genotype.

By the way the haplotypes are sorted, each block starts long at the "top" and gets shorter as haplotypes begin to differ in sort order until finally the match grows too short. If we consider only those match blocks with two or more genotypes, then we can see that all genotype substrings from where the PBWT has rotated the sequences to, to the length of the shortest match, are connected in the Clark Consistency Graph. For example, in a match block consisting of

```
20200000000000010000110001100001000111010001100110000000000000001000
22000000000000020020220001100002000122220002220222000000000020000
22000000000002000010000001100002020
2200000000002020002000002
```

Where the bottommost match begins at index $i$ and ends at index $j$. Each genotype can share a partial haplotype with the genotype below, so we define the Clark Consistency Graph on the substrings beginning at $i$ and ending at $j$,

```
20200000000000010000110001
22000000000000020020220001
```

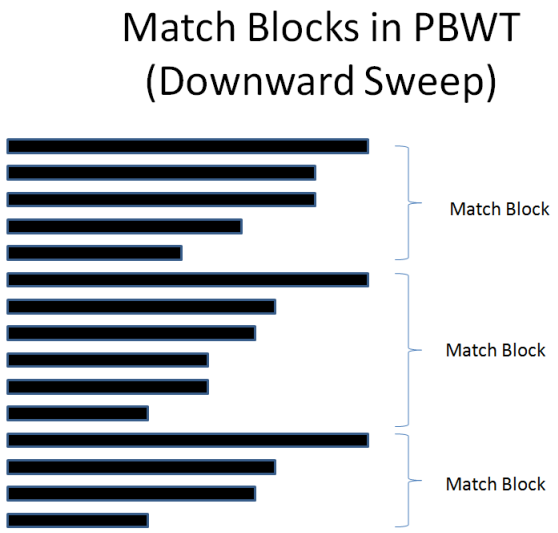



Figure 9: An abstraction of the shortening pattern of match blocks

```
22000000000002000010000001
22000000000020200020000002
```

and they are connected, as desired. Since we can draw an edge between the 2nd haplotype and the 4th,

```
22000000000000020020220001
22000000000020200020000002
```

because they can share the haplotype

```
11000000000000000010000001
```

then we must be able to draw that edge between all haplotypes between them, so this comprises a clique. Inspection shows that this haplotype can indeed be shared between the second and 3rd, and the 3rd and 4th genotypes.

I further conjecture that there is an equivalence between the PBWT, an algorithm on suffix arrays, and Tractatus, an algorithm on suffix trees, which would imply that solving a genotype tract finding problem on one of those structures would provide a solution to solving a genotype tract finding problem on the other structure.

## 5.3 Algorithms leveraging Tract Information

Computationally efficient tract finding is currently an active field of research. Browning and Browning have recently released a version of BEAGLE that performs haplotype imputation which scales to over a million of reference genomes.

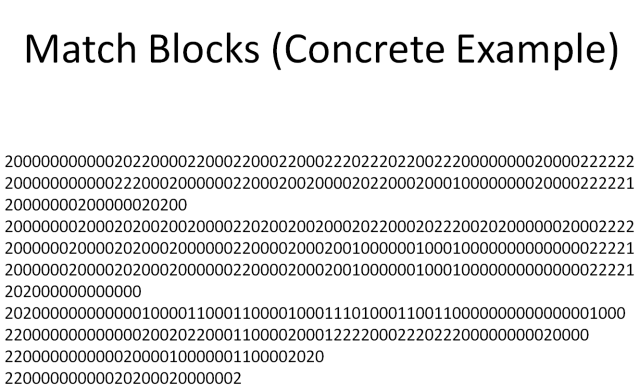



Figure 10: An example of matchblocks, using simulated genotype data

[Browning and Browning, 2016] This uses a different clustering model based on the Li Stephens framework - the imputation problem is on a haploid HMM rather than a diploid HMM, so there are significantly fewer states in the haploid HMM as compared to the diploid HMM.

In addition to this, an algorithm using the PBWT was recently been proposed for inferring diploid haplotypes, also in the Li Stephens framework, treating each state as a tract that generates a sequence of observations rather than a single allele. This algorithm provides implementations of both a viterbi-like algorithm and what is termed a FastLS (Fast Li Stephens) which has a runtime that is constant in the number of haplotypes in the reference panel. [Lunter, 2016]

# 6 Conclusion

HapROSE was written with the intent of serving as both a way to bring multiple implementations of haplotype phasing "under one roof", so to speak, so that we could perform benchmarking by measuring the properties and behaviors of different heuristics. In particular, we looked at how the Browning Similar function produced different learned APFA than the Ron Similar function at lower amounts of reference data, and discussed whether Browning's improved speed came at a particularly high cost when it came to accuracy. Since HapROSE is all in a single software package, the effects of the heuristics on speed and accuracycould be studied and compared in a controlled environment where the same optimizations are performed for each heuristic.

The major findings of this paper are that the BEAGLE similarity heuristic allows the stochastic EM algorithm to perform more accurate phasing than the Ron et. al. heuristic when they are tuned to run for roughly the same period of time, but the Ron outperforms BEAGLE at tighter values of its distance parameter at the cost of increased runtime.

# 7 Future Directions

## 7.1 Faster Phasing

The algorithms studied here would benefit greatly from the introduction of more information to help guide the inference being performed. In particular, knowing what sorts of tracts appear in the reference haplotypes will allow for faster training of the models, and knowing how those tracts correlate over longer ranges can help overcome the limitations of assuming that allele sequences possess the Markov property - something that helps to greatly speed up the inference, but ultimately sacrifices accuracy since any long-term correlations between haplotype blocks are lost.

In terms of ways to improve the speed of a phasing algorithm, it is unnecessary to perform the entire forward-backward and viterbi algorithm on the entire length of the genotype to be phased when, in reality, we are only interested in a small local subset of frequent haplotypes. The entire model is so efficient precisely because we are leveraging the Markov property, so we may be able to discard potential noise from far-away segments of the genome that propagate throughout the chain during the message passing steps of the forward backward algorithm simply by performing inference on a structure with a given max depth. It may be more prudent, and similarly accurate, to train a model with limited depth that was able to capture every haplotype block, while itself only being a fraction of the length of the whole sequence.

To this end, it may be possible to adapt the Predictive Suffix Tree to this purpose [Ron et al., 1996]. This data structure, defined in depth in another paper by Ron et. al., takes in a prefix of a suffix (a substring) and provides a next symbol probability, over all symbols in the alphabet.

It is conceivable to have one such tree of adequate depth predicting the next symbol in the forward direction, and another such tree of adequate depth predicting the next symbol in the backward direction. Since we have algorithms that construct suffix trees in linear time, there may be a way to construct and perform inference of similar quality much faster using this data structure.

## 7.2 More Accurate Phasing

To provide a more accurate phasing, we have to go the other way. Rather than choosing to forget more to speed up phasing, we need to introduce more information about the haplotypes and genotypes. A natural next step for this research is to augment the existing implementation to leverage the output of HapCompass in order to guide the training of the model such that more weight can be assigned to those edges that correspond to observed phased pairs of alleles in short regions.

Another excellent resource for improving phase estimates are the long reads produced by Pacific Biosciences (PacBio)[Biosciences, 2016], which can reach over 20k kilobases in length. Data of this kind is a unique source of long-range phasing information, since it will help us to find more correlated SNPs that may

be separated by long distances on the genome. These longer correlations are difficult to detect, and may even be completely lost when training the model. To incorporate this sort of information into an inference algorithm would require a thorough understanding of the PacBio read error model, but it can be used to strongly influence either the model building phase or the calculation of the posterior distribution of the diploid HMM.

# References

[Bansal et al., 2008] Bansal, V., Halpern, A. L., Axelrod, N., and Bafna, V. (2008). An mcmc algorithm for haplotype assembly from whole-genome sequence data. *Genome research*, 18(8):1336–1346.

[Biosciences, 2016] Biosciences, P. (2016). Smrt sequencing: Read lengths. [Online; accessed 14-April-2016].

[Browning and Browning, 2016] Browning, B. and Browning, S. (2016). Genotype imputation with millions of reference samples. *The American Journal of Human Genetics*, 98(1):116 – 126.

[Browning and Browning, 2007a] Browning, B. L. and Browning, S. R. (2007a). Efficient multilocus association testing for whole genome association studies using localized haplotype clustering. *Genetic Epidemiology*, 31(5):365–375.

[Browning and Browning, 2007b] Browning, S. R. and Browning, B. L. (2007b). 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(5):1084 – 1097.

[Browning and Browning, 2011] Browning, S. R. and Browning, B. L. (2011). Haplotype phasing: existing methods and new developments. *Nat. Rev. Genet.*, 12(10):703–714.

[Durbin, 2014] Durbin, R. (2014). Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272.

[Gusfield, 2000] Gusfield, D. (2000). A practical algorithm for optimal inference of haplotypes from diploid populations. In *ISMB*, pages 183–189.

[Halldorsson et al., 2011] Halldorsson, B. V., Aguiar, D., Tarpine, R., and Istrail, S. (2011). The Clark phaseable sample size problem: long-range phasing and loss of heterozygosity in GWAS. *J. Comput. Biol.*, 18(3):323–333.

[Halldorsson et al., 2004] Halldorsson, B. V., Bafna, V., Edwards, N., Yooseph, S., and Istrail, S. (2004). A survey of computational methods for determining haplotypes. In *Lecture Notes in Computer Science (2983): Computational Methods for SNPs and Haplotype Inference*, pages 26–47. Springer.

[Howie et al., 2009] Howie, B. N., Donnelly, P., and Marchini, J. (2009). A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. *PLoS Genet*, 5(6):e1000529.

[Hudson, 2004] Hudson, R. R. (2004). ms a program for generating samples under neutral models.

[Li and Stephens, 2003] Li, N. and Stephens, M. (2003). Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4):2213–2233.

[Lin et al., 2002] Lin, S., Cutler, D. J., Zwick, M. E., and Chakravarti, A. (2002). Haplotype inference in random population samples. *The American Journal of Human Genetics*, 71(5):1129–1137.

[Loh et al., 2015] Loh, P.-R., Palamara, P. F., and Price, A. L. (2015). Fast and accurate long-range phasing and imputation in a uk biobank cohort. *bioRxiv*, page 028282.

[Lunter, 2016] Lunter, G. (2016). Fast haplotype matching in very large cohorts using the li and stephens model. *bioRxiv*.

[Niu et al., 2002] Niu, T., Qin, Z. S., Xu, X., and Liu, J. S. (2002). Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *The American Journal of Human Genetics*, 70(1):157–169.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. In *PROCEEDINGS OF THE IEEE*, pages 257–286.

[Ron et al., 1995] Ron, D., Singer, Y., and Tishby, N. (1995). On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of the eighth annual conference on Computational learning theory*, pages 31–40. ACM.

[Ron et al., 1996] Ron, D., Singer, Y., and Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine learning*, 25(2-3):117–149.

[Scheet and Stephens, 2006] Scheet, P. and Stephens, M. (2006). A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase. *The American Journal of Human Genetics*, 78(4):629–644.

[Stephens and Scheet, 2005] Stephens, M. and Scheet, P. (2005). Accounting for decay of linkage disequilibrium in haplotype inference and missing-data imputation. *The American Journal of Human Genetics*, 76(3):449–462.