Supplementary Information for "Reference-based phasing using the Haplotype Reference Consortium panel"

Po-Ru Loh, Petr Danecek, Pier Francesco Palamara, Christian Fuchsberger, Yakir A Reshef, Hilary K Finucane, Sebastian Schoenherr, Lukas Forer, Shane McCarthy, Goncalo R Abecasis, Richard Durbin, Alkes L Price

Contents

Nature Genetics: doi:10.1038/ng.3679

Nature Genetics: doi:10.1038/ng.3679

Supplementary Note: Eagle2 algorithm

1 Background

1.1 Existing phasing methods

To our knowledge, all general-purpose methods for genome-wide population-based phasing that have enjoyed widespread use in the past decade apply or innovate on the Li-Stephens haplotype-copying HMM [21]. Very simply, given a set of K reference haplotypes, the Li-Stephens model generates a new haplotype as an imperfect mosaic of reference haplotype segments. The states of the model (K states per SNP) designate reference haplotypes, and the Markov process moves left-to-right across a chromosome, choosing at each successive SNP whether to stay on the current reference haplotype or jump to a new haplotype (the transitions) and whether to emit the reference haplotype's allele or a mutant allele (the emissions). Thus, the Li-Stephens HMM has the virtues of (approximately) modeling haplotype frequencies, mutation, and recombination. Moreover, it naturally extends to a generative model for diplotypes (i.e., pairs of haplotypes) assuming independence of maternally- and paternally-derived haplotypes [5].

The diploid version of the Li-Stephens model is still Markovian and hence lends itself to phase inference (for a given sample conditional on either a phased reference panel or estimated haplotypes for other samples) via forward-backward, Viterbi, or Gibbs sampling approaches. The model was first used (very successfully) for phasing in the PHASE v2 algorithm [5]. However, as data sizes increased, the HMM computations for this model quickly became intractable; even with recursive computation, calculation of forward and backward probabilities requires $O(MK^2)$ time for M SNPs and K reference haplotypes [6]. This computational bottleneck led to a series of innovations over the past decade, all of which involve some type of approximation:

- fastPHASE [6] performs clustering on haplotypes and approximates each cluster with a single state, reducing K from the number of reference haplotypes to the number of clusters.
- Beagle [7, 9] applies a localized haplotype clustering approach [32, 33], reducing the state space by approximating the HMM as a haplotype graph and carefully estimating transition probabilities between clusters.
- HAPI-UR [11] operates on windows of dozens of SNPs at a time, collapsing haplotypes identical within a window into a single state (thus approximating the HMM), and discarding states inconsistent with the diploid individual being phased.
- SHAPEIT [10, 12, 14] conditions on only a subset of K locally best reference haplotypes (in version 2 and higher), building an individual-specific HMM on 3-het haplotype clusters and

Nature Genetics: doi:10.1038/ng.3679

performing Gibbs sampling in $O(MK)$ time per MCMC iteration.

- Eagle1 [13] (in its final HMM-like step) analyzes a rudimentary HMM, using a fast scoring scheme to search for an approximate Viterbi path through local subsets of K haplotypes and applying a branching-and-pruning beam search to limit computational cost to $O(MKP)$, where P is the beam width (i.e., the number of haplotype pairs retained after pruning).

Our recent work on Eagle1 [13] aimed to improve phasing speed at very large sample sizes by using a hybrid approach of long-range phasing [8] followed by approximate HMM-based phasing. (We note that long-range phasing alone suffices—and in fact may yield better accuracy than HMM-based inference due to the challenge of resolving conflicts in IBD information—when a sizable fraction of a population has been genotyped; however, this threshold has only been reached in Iceland [8].) We demonstrated that when used to phase N=150K UK Biobank samples, Eagle1 achieved 1–2 order of magnitude speedups over existing methods at equal or greater accuracy. However, for $N<$50K (and for underrepresented ethnicities in larger data sets), we observed that Eagle1 achieved suboptimal accuracy, as expected given its makeshift HMM: our HMM approximation was good enough to quickly identify strong phase evidence from sharing of long haplotypes, but not good enough to properly evaluate regions with less-certain phase.

1.2 New approach: Eagle2

Given the above observation, we were motivated to develop a completely new statistical phasing algorithm, Eagle2, that can attain the speed of Eagle1 without losing accuracy at smaller sample sizes. Unlike Eagle1, Eagle2 analyzes a full probabilistic model similar to the diploid Li-Stephens model used by existing methods (but relaxing the Markovian assumption). The key difference between Eagle2 and previous methods is that whereas previous approaches *approximate the haplotype structure* (e.g., by merging haplotypes into local clusters) to produce a more tractable HMM, Eagle2 (1) *efficiently represents the full haplotype structure* in a way that losslessly condenses locally matching haplotypes; and (2) using this representation, *selectively explores the diplotype space* in a way that only expends computation on the most likely phase paths (i.e., diplotypes with highest posterior probabilities). We achieve (1) by introducing a new data structure, the HapHedge, which requires only $O(MK)$ time to generate using the positional Burrows-Wheeler transform (PBWT) [20], and we achieve (2) by using a branching-and-pruning beam search (similar in spirit to the beam search used in Eagle1 [13] but very different in implementation).

Intuitively, the main advantage of performing approximate inference by limiting exploration of the diplotype space vs. simplifying the model itself is that the first approach (which we adopt in Eagle2) has the potential to lose little or no accuracy if most phase paths are extremely unlikely. The same can be said for the usual approach of reducing model complexity *only if* the model is

Nature Genetics: doi:10.1038/ng.3679

simplified in a manner that eliminates unlikely phase paths without accidentally losing (or damping) likely phase paths. SHAPEIT2 [12] seems to achieve this behavior by building its HMMs using small sets of locally-best references selected in an individual-specific manner, but in general, simplifying the HMM in a way that does not lose accuracy appears to be quite challenging (based on our benchmarks against existing methods). In particular, methods that create a single simplifed HMM by locally clustering haplotypes and then modeling transition probabilities between consecutive clusters necessarily lose longer-range information.

An additional feature of the Eagle2 approach is that it no longer relies on standard forward-backward HMM recursions for computational tractability, and as such, allows us to introduce non-Markovian recombination probabilities that more accurately model the coalescent process. Specifically, as we move along the genome extending a shared haplotype, the probability of recombining off the shared haplotype (conditional on the length shared so far) should decrease as the sharing length grows. HMMs are unable to model this history-dependent behavior (unless the state space is augmented with an age parameter at the expense of increased model complexity), whereas Eagle2 has no trouble doing so. That said, we believe that this feature of Eagle2 is currently only of theoretical interest; our benchmarks indicate that Eagle2's performance is insensitive to the recombination model (Supplementary Table 11).

To be fair, the Eagle2 approach of exploring only a small portion of the diplotype space does have the theoretical downside of being heuristic: by choosing to consider only part of the space, we are forced to adopt less-rigorous approximate inference procedures and lose the theoretical guarantees of HMM posterior decoding or sampling. Having to make this trade-off is not ideal, but given our empirical evidence supporting the accuracy of Eagle2, its gains in speed over existing methods, and the fact that other methods make theoretically challenging approximations at the level of HMM-building, we believe the trade-off is worthwhile.

Thus far we have compared methods based on the core algorithm that they use to phase a single sample using a given set of reference haplotypes. In practice, input for phasing software typically contains many samples to be phased and may or may not include a phased reference panel. Procedures for handling these situations are fairly straightforward; e.g., algorithms typically iterate over samples, (re-)estimating haplotypes for each sample using the current haplotype estimates for all other samples along with any provided reference haplotypes. The precise top-level procedure that our software uses when analyzing a many-sample data set is described in Online Methods. In the remainder of this note, we stay one level lower and detail the core subroutines that Eagle2 applies to phase a single sample given a reference set and to impute missing genotypes in the same scenario. We then detail the core HapHedge data structure that makes these algorithms possible.

Nature Genetics: doi:10.1038/ng.3679

2 Eagle2 core phasing algorithm

Here we describe the mathematical model and core algorithm that Eagle2 uses to phase a single sample given a set of reference haplotypes.

2.1 Diplotype probability model

We consider a single "target" sample with diploid genotypes $g_1, \ldots, g_M \in \{0, 1, 2\}$ at M biallelic markers. (We note that multiallelic markers can be coded as multiple biallelic markers.) We wish to model the distribution over possible diplotypes—i.e., pairs of maternally- and paternally-derived haplotypes $h_1^\alpha, \ldots, h_M^\alpha \in \{0, 1\}$ and $h_1^\beta, \ldots, h_M^\beta \in \{0, 1\}$—that could underlie the observed genotypes $g_1, \ldots, g_M$, taking into account population haplotype structure, recombination, and the possibility of single-SNP discrepancies (due to genotyping error, mutation, or gene conversion, any of which could result in $g_m \neq h_m^\alpha + h_m^\beta$). We assume we are given a set of K (phased) reference haplotypes $\{h_1^k, \ldots, h_M^k\}_{k=1}^K$ with which to model population haplotype structure. For notational convenience, we will use colons to abbreviate lists, e.g., $g_{1:M}$ to abbreviate $g_1, \ldots, g_M$, from here on.

To model each of the above features of haplotype distributions, we use the general haplotype-copying-with-error approach. Specifically,

- we put a prior $P(h_{1:M}^\alpha, h_{1:M}^\beta)$ on diplotypes $(h_{1:M}^\alpha, h_{1:M}^\beta)$ based on the process of copying reference haplotypes with recombination,
- we put a distribution $P(g_{1:M} \mid h_{1:M}^\alpha, h_{1:M}^\beta)$ on observed genotypes given putative maternally- and paternally-derived haplotypes (to model single-SNP discrepancies), and
- we infer (relative) posterior probabilities $P(h_{1:M}^\alpha, h_{1:M}^\beta \mid g_{1:M})$ using Bayes' rule:

$$P(h_{1:M}^\alpha, h_{1:M}^\beta \mid g_{1:M}) \propto P(g_{1:M} \mid h_{1:M}^\alpha, h_{1:M}^\beta) \cdot P(h_{1:M}^\alpha, h_{1:M}^\beta). \quad (1)$$

We now define the probabilities $P(h_{1:M}^\alpha, h_{1:M}^\beta)$ and $P(g_{1:M} \mid h_{1:M}^\alpha, h_{1:M}^\beta)$.

2.1.1 Distribution of recombined haplotypes given reference haplotypes

To write down a prior $P(h_{1:M}^\alpha, h_{1:M}^\beta)$ on diplotypes, we first make the independence approximation

$$P(h_{1:M}^\alpha, h_{1:M}^\beta) \approx P(h_{1:M}^\alpha) \cdot P(h_{1:M}^\beta), \quad (2)$$

where the probabilities $P(h_{1:M}^\alpha)$ and $P(h_{1:M}^\beta)$ for the maternally- and paternally-derived haplotypes come from the same distribution on haplotypes $h_{1:M}$. (We will thus be unable to distinguish the maternally- and paternally-derived haplotypes; all distributions will be symmetric with respect to $h_{1:M}^\alpha$ and $h_{1:M}^\beta$.)

Nature Genetics: doi:10.1038/ng.3679

For a single haplotype $h_{1:M}$, we model $h_{1:M}$ as a mosaic of recombined haplotype segments $h_{1:A}, h_{A+1:B}, \ldots, h_{L+1:M}$ copied from the reference haplotypes $\{h^k_{1:M}\}_{k=1}^K$. (We have denoted the recombination breakpoints partitioning the sites $1{:}M$ using the notation $A, B, \ldots, L$—which should be taken to represent a list of arbitrary length—to avoid double subscripts.) We may thus decompose $P(h_{1:M})$ as a sum over possible sets of recombination breakpoints $A, B, \ldots, L$:

$$\begin{aligned} P(h_{1:M}) &= \sum_{A,B,\ldots,L} f(h_{1:A}) \cdot P(\text{rec } A \mid \text{rec } 0) \cdot f(h_{A+1:B}) \cdot P(\text{rec } B \mid \text{rec } A) \\ &\qquad \cdots f(h_{L+1:M}) \cdot P(\text{rec } M \mid \text{rec } L), \end{aligned} \tag{3}$$

where

$$f(h_{X+1:Y}) = \frac{\#\{k : h^k_{X+1:Y} = h_{X+1:Y}\}}{K} \tag{4}$$

denotes the frequency of the haplotype segment $h_{X+1:Y}$ in the reference and $P(\text{rec } Y \mid \text{rec } X)$ denotes the probability that the next recombination occurs between markers Y and $Y+1$ given that the previous recombination occurred between markers X and $X+1$.

For simplicity, we make the approximation that the previous recombination occurred exactly at marker X, and we let u denote the genetic distance from X to Y and v denote the genetic distance from X to $Y+1$. Then $P(\text{rec } Y \mid \text{rec } X)$ is just the integral of the IBD length distribution—i.e., the distribution of lengths of most recent shared tracts between a given haplotype and our set of K reference haplotypes—from u to v. (We note that in the context of typical HMM transition probabilities, this recombination probability accumulates the probabilities of no transition from X to Y together with the probability of a transition between Y and $Y+1$.)

We approximate this distribution under the coalescent-based SMC model [34], obtaining the following form (given in ref. [35]):

$$P(\text{rec } Y \mid \text{rec } X) = \frac{1}{(1+u/a)^2} - \frac{1}{(1+v/a)^2}, \tag{5}$$

where a is a parameter specifying the expected IBD length. We provide a full discussion of this formula and its relationship to the traditional Li-Stephens model in Section 4.

2.1.2 Distribution of genotypes given a diplotype

To model single-SNP discrepancies between a diplotype $h^\alpha_{1:M}, h^\beta_{1:M}$ and the observed genotypes $g_{1:M}$, we use the following simple approximation:

$$P(g_{1:M} \mid h^\alpha_{1:M}, h^\beta_{1:M}) \propto \epsilon^{n_{\text{err}}}, \tag{6}$$

Nature Genetics: doi:10.1038/ng.3679

where ϵ is an "error probability" parameter that roughly represents the chance of an allele modification, and

$$n_{\text{err}} = \sum_{m=1}^{M} |g_m - (h_m^\alpha + h_m^\beta)| \tag{7}$$

is the total number of allele modifications required to transform the diplotype $(h_{1:M}^\alpha, h_{1:M}^\beta)$ to match the observed genotypes $g_{1:M}$. We note that this intuition does not precisely produce equation (6): strictly speaking, we should have an extra factor of 2 when $|g_m - (h_m^\alpha + h_m^\beta)| = 1$ (because the discrepancy could occur on either haplotype), and we should also have a term $(1-\epsilon)^{2M-n_{\text{err}}}$ for non-error probabilities (which we have instead approximated to 1). However, in practice, we have observed that our inference is insensitive to the value of ϵ in the range 0.0003–0.01 (Supplementary Table 12)—and in reality, the distributions of genotyping errors, mutations, and gene conversions are much more complex anyway (see next paragraph)—so we adopt the approximation in equation (6) due to its simplicity.

Fleshing out the above point, we note that because our copying model for $h_{1:M}$ as a mosaic of segments $h_{X+1:Y}$ does not allow for mutation, the probability $P(g_{1:M} \mid h_{1:M}^\alpha, h_{1:M}^\beta)$ must actually account for any mutations or gene conversions that occur along the lineages connecting copied haplotype segments to the target sample. (To be completely precise, in the model we have defined, $h_{1:M}^\alpha, h_{1:M}^\beta$ are actually best described as "mosaics of surrogate maternal and paternal haplotypes"—i.e., composites of haplotypes shared with (distant) relatives of the mother and father of the target sample—rather than the maternally- and paternally-derived haplotypes of the target sample.) In theory, we could therefore improve the accuracy of our model by allowing the error parameter ϵ to be a function of haplotype copying length (which scales inversely with time to coalescence and thus with accumulated mutation rate along a lineage). However, for large reference panels, in which coalescences between target haplotypes and closest reference haplotypes are relatively recent, single-SNP discrepancies are most often produced by genotyping error, so we ignore this subtlety and use a fixed value of ϵ.

2.2 Algorithm overview: Fast computation and focused search

The Eagle2 phasing algorithm applies two main ideas to perform fast and accurate phase inference given the probabilistic model we have just described. The first idea is a new approach to efficient computation of haplotype probabilities under a copying model (in our case, equation (3)). Naïvely, these computations require exponential time. The standard approach to overcoming this barrier is to observe that within a K_{HMM}-state HMM, recursion allows computation of *all* marginal probabilities (for all K_{HMM} states at each of M positions) in $O(MK_{\text{HMM}}^2)$ time. In Eagle2, we take a completely different recursive approach that computes the probability of a *single* haplotype $P(h_{1:M})$ in $O(M)$ time—independent of the number of reference haplotypes K—after creation of

Nature Genetics: doi:10.1038/ng.3679

a new data structure, the HapHedge, in $O(MK)$ time. The HapHedge essentially consolidates all reference haplotypes sharing a common prefix (starting at any given position) into a single atom of data, thus eliminating future computation that scales with K.

Of course, being able to very rapidly compute the probability of a single haplotype is only useful if we can identify a small subset of haplotype probabilities that are worth computing; to this end, we need a second key idea. We perform a beam search from left to right across the genome, propagating a small set of P likely diplotypes that ideally represent most of the posterior probability mass in the full diplotype space. This approach essentially focuses computational effort on a small subset of the diplotype space (vs. expending computation evenly across the space as in HMM recursion), which is advantageous when most of the space is unlikely but difficult to discard *a priori*.

We now expand on each of these ideas; we fully describe the haplotype probability computation (aside from specification of the HapHedge, which we defer to Section 3), and we provide further details and intuition for the beam search, which we formally treat in Section 2.3.

2.2.1 Efficiently computing haplotype probabilities

From equation (3), we wish to compute

$$P(h_{1:M}) = \sum_{A,B,\ldots,L} f(h_{1:A}) \cdot P(\text{rec } A \mid 0) \cdot f(h_{A+1:B}) \cdot P(\text{rec } B \mid A) \cdots f(h_{L+1:M}) \cdot P(\text{rec } M \mid L). \tag{8}$$

It turns out that each factor of each summand in the above formula can be computed in $O(1)$ time. This statement is obvious for the recombination probabilities $P(\text{rec } Y \mid \text{rec } X)$ (abbreviated as $P(\text{rec } Y \mid X)$ in the above), which are given by equation (5). As for the the haplotype segment frequencies $f(h_{X+1:Y})$, the HapHedge data structure precisely supports these lookups in constant time (assuming we have saved state information from previous lookups of the sub-segments $h_{X+1:Y-1}$, which is always the case for our algorithm).

The only remaining obstacle is the exponential sum over sets of recombination breakpoints $A, B, \ldots, L$, but conveniently, equation (8) naturally admits the recursion

$$P(h_{1:Y}) = \sum_{X=1}^{Y-1} P(h_{1:X}) \cdot f(h_{X+1:Y}) \cdot P(\text{rec } Y \mid X). \tag{9}$$

This recursion yields an $O(M^2)$ dynamic programming algorithm for computing $P(h_{1:M})$.

We further reduce running time by restricting the sum over previous recombination points X

Nature Genetics: doi:10.1038/ng.3679

to sites within a distance H of the current site Y:

$$P(h_{1:Y}) \approx \sum_{X=Y-H}^{Y-1} P(h_{1:X}) \cdot f(h_{X+1:Y}) \cdot P(\text{rec } Y \mid X). \tag{10}$$

That is, we allow the model to "forget" history older than H positions earlier in the genome, or equivalently, we limit the maximum length of copied haplotypes to H. This approximation reduces the cost of (approximately) computing $P(h_{1:M})$ to $O(MH)$. We set H to be the distance spanned by 100 heterozygous sites in the target sample (corresponding to a few cM at the density of typical genotyping arrays) or 1cM in very dense data; increasing H beyond this value appears to have little effect (Supplementary Tables 10 and 13). (In Section 2.4, we will describe optimizations that condense sites between consecutive target hets, hence the choice of this positional unit.)

2.2.2 Exploring only the likeliest diplotypes

The computational tricks above allow $O(MH)$-time estimation of $P(h_{1:M})$ for any $h_{1:M}$ and thus also $P(h^{\alpha}_{1:M}, h^{\beta}_{1:M})$ and the desired posterior $P(h^{\alpha}_{1:M}, h^{\beta}_{1:M} \mid g_{1:M})$ for any given diplotype. We are left with the challenge of determining which diplotypes to evaluate, which we treat as a search problem. That is, we build diplotypes left-to-right across a chromosome, and at each successive marker, for each diplotype, we consider extending each of its haplotypes to either the 0 or 1 allele at the next marker m. This branching procedure increases the number of diplotypes under consideration by a factor of 4, so to prevent an exponential explosion, we then prune the set of diplotypes back down to at most P diplotypes $\{(h^{\alpha(p)}_{1:m}, h^{\beta(p)}_{1:m})\}_{p=1}^{P}$, keeping the most likely diplotypes according to posterior probabilities $P(h^{\alpha}_{1:m}, h^{\beta}_{1:m} \mid g_{1:m})$ based on information thus far (i.e., markers 1:m). The total cost of the entire procedure is $O(MHP)$. The intuition behind the beam search is that it greedily prioritizes likely diplotypes, and while future information may show that some of these diplotypes have only low-probability extensions, the hope is that the beam will be wide enough (i.e., contain enough diplotypes) to be robust to such local maxima.

The procedure that we have described is a good start, but we should be a bit skeptical: How can we expect a small number of greedily-selected diplotypes to adequately represent the exponential space of phase paths, especially at the scale of entire chromosomes? Even if the posterior distribution on diplotypes is locally well-concentrated, the space will inevitably grow exponentially, which causes two opposite types of problems for our beam search: parallelism and degeneracy.

- **Pitfall 1: Parallelism.** This problem amounts to a lack of beam diversity in the recent past (i.e., when restricted to the preceding local region of the genome): the beam can fragment into several paths with similar probabilities and then follow each path along parallel trajectories (i.e., identical phase paths after the split). This behavior is in fact highly likely if the search space is locally well-constrained (as it tends to be), and it is problematic because

Nature Genetics: doi:10.1038/ng.3679

fragmentation of the beam into parallel paths effectively reduces its width and thus its ability to overcome local maxima.

- **Pitfall 2: Degeneracy.** This problem is a lack of beam diversity in the more distant past. Our greedy search prioritizes best paths in an exponentially growing space, pruning away less-likely paths; as such, the set of P diplotypes that comprise the beam front at any given position will (hopefully) be a good representation of the diversity of paths through recent positions. However, if we trace these P diplotypes back toward the start of the chromosome, they will inevitably collapse into a single trajectory, providing a poor representation of the marginal posterior on diplotypes at positions far behind the current position.

To overcome these pitfalls, we need to exercise greater finesse as we conduct our beam search and perform phase inference. We make three main adjustments:

- **Solution 1: Merging.** To deal with parallelism, when we observe that two diplotypes under consideration agree at the last 64 target hets, we merge the diplotypes (by pruning one diplotype and adding its "weight" to the other diplotype; we defer a precise definition of this procedure to the formal exposition in Section 2.3).

- **Solution 2: Fixed-lag smoothing.** To deal with degeneracy, we do not attempt to use the final beam front of full diplotypes $(h^{\alpha}_{1:M}, h^{\beta}_{1:M})$ to call phase throughout the chromosome. Instead, we call the relative phase of a pair of consecutive hets using only a limited amount of future (forward) information, a procedure similar to fixed-lag smoothing from signal processing. Specifically, at any position m, we use the P diplotypes $\{(h^{\alpha(p)}_{1:m}, h^{\beta(p)}_{1:m})\}_{p=1}^{P}$ in the beam front to call phase at a distance Δ=20 hets earlier (by examining the marginal posterior distribution at the 19th- and 20th-most recent hets). Effectively, this choice means that at any position m, we need the beam front to only be a good approximation of the marginal posterior over "recent history" in the interval $[m - \Delta, m]$.

- **Solution 3: Refinement via constrained search.** Finally, to improve the ability of the beam search to overcome local maxima, we apply a two-step search. The first step is to run the beam search described above; however, rather than simply making phase calls during this run, we also estimate marginal posteriors for each consecutive pair of hets, thus obtaining phase confidences. We then rank the phase confidences from most to least confident, identify high-confidence phase calls, and run a second beam search in which we constrain phase at high-confidence het pairs to match the first-iteration calls. The idea is that these phase constraints propagate information backward along the chromosome, allowing the second-iteration search better able to avoid obstacles while focusing attention on only the most difficult phase calls.

Nature Genetics: doi:10.1038/ng.3679

2.2.3 Computational cost

The total computational cost of the algorithm we have described for beam propagation and phase inference algorithm is $O(MHP)$, where M is the number of markers, H is the history parameter governing maximum copying length in the mosaic model, and P is the number of paths in the beam front. We also need to take into account the $O(MK)$ cost of building the HapHedge data structure on K reference haplotypes—but if we were to implement the exact algorithm described thus far, this data structure would only need to be built once (rather than once per target individual).

Because the constant factor on HapHedge construction is quite small, we have chosen to instead build individual-specific condensed HapHedge structures on subsets of $K = 10,000$ best reference haplotypes per individual; doing so has the advantage of allowing the beam search to proceed one het at a time rather than one marker at a time, thus reducing the constant factor on the $O(MHP)$ term. Given a panel of $2N$ reference haplotypes as input, our algorithm for selecting the best K reference haplotypes for a given target sample currently runs in $O(MN)$ time with a very small constant factor via bit-level parallelism; thus, the overall procedure takes $O(MN + MHP)$ time with small constant factors on both terms. Details of these optimizations are given in Section 2.4.

2.3 Formal inference procedure

We now present a formal exposition of our beam search-based inference procedure that implements the preceding intuition.

2.3.1 Approximating marginal posteriors using ensembles of diplotypes

For each $m = 1, \ldots, M$, we wish to create an ensemble of $\leq P$ weighted diplotypes, denoted $\{(w^{(p)}, h_{1:m}^{\alpha(p)}, h_{1:m}^{\beta(p)})\}_p$, with the property that the truncation of the ensemble to the "recent past" $[m - \Delta, m]$ provides a good approximation of the true marginal posterior on diplotypes given genotypes $g_{1:m}$ "seen so far." That is, we want:

$$\begin{aligned} & P(h_{m-\Delta:m}^{\alpha}, h_{m-\Delta:m}^{\beta} \mid g_{1:m}) \\ & \propto \sum_p w^{(p)} \cdot \hat{P}(h_{1:m}^{\alpha(p)}, h_{1:m}^{\beta(p)} \mid g_{1:m}) \cdot \mathbf{1}(h_{m-\Delta:m}^{\alpha(p)}, h_{m-\Delta:m}^{\beta(p)} = h_{m-\Delta:m}^{\alpha}, h_{m-\Delta:m}^{\beta}), \end{aligned} \tag{11}$$

where the $\hat{P}$ term indicates the approximate posterior we described how to compute in equations (1), (2), (6), and (10), and $\mathbf{1}(\cdot)$ denotes the indicator function. (We could have absorbed these terms into the weights $w^{(p)}$, but we wish to view the weights as diplotype "multiplicities," which will be intuitive when we later discuss merging diplotypes.)

Is there hope of achieving the approximation suggested in equation (11) using only a small ensemble of diplotypes? The feasibility of this task is essentially a question of whether or not the

posterior $P(h^{\alpha}_{m-\Delta:m}, h^{\beta}_{m-\Delta:m} \mid g_{1:m})$ is concentrated on a small support—which should be the case if phase is relatively well-constrained over the interval $[m-\Delta, m]$ (i.e., only a few phase paths through this interval are likely).

2.3.2 Propagating diplotypes: Branching, merging, pruning, and calling

We construct the ensembles $\{(w^{(p)}, h^{\alpha(p)}_{1:m}, h^{\beta(p)}_{1:m})\}_p$ for $m = 1, \ldots, M$ by propagating a "beam front" across the chromosome. We initialize the beam front at $m = 0$ with a single diplotype on no markers (containing no haplotype information) and proceed left to right, creating beam front m (i.e., the m-th ensemble) from beam front $m-1$ according to the following procedure.

Branching. For each diplotype $\{(w^{(p')}, h^{\alpha(p')}_{1:m-1}, h^{\beta(p')}_{1:m-1})\}_{p'}$ in ensemble $m-1$ (indexed by p'), we create its four possible extensions by appending $h^{\alpha}_m \in \{0,1\}$ and $h^{\beta}_m \in \{0,1\}$ to $h^{\alpha(p')}_{1:m-1}$ and $h^{\beta(p')}_{1:m-1}$. (We consider all four possible extensions to allow for single-SNP discrepancies due to genotyping error, etc.) We give its weight $w^{(p')}$ to all four possible extensions (for now), and we compute the posterior probabilities $\hat{P}(h^{\alpha(p')}_{1:m-1}h^{\alpha}_m, h^{\beta(p')}_{1:m-1}h^{\beta}_m \mid g_{1:m})$ in $O(H)$ time by making use of the recursion in equation (10), looking up previously computed values (i.e., applying dynamic programming).

Merging and pruning. We now have up to $4P$ diplotypes on $1{:}m$ and need to reduce to an ensemble of size $\leq P$. To do so, we employ two techniques: merging and pruning. First, if two weighted diplotypes $(w^{(p)}, h^{\alpha(p)}_{1:m}, h^{\beta(p)}_{1:m})$ and $(w^{(q)}, h^{\alpha(q)}_{1:m}, h^{\beta(q)}_{1:m})$ are identical on the interval $[m-\Delta, m]$, i.e.,

$$h^{\alpha(p)}_{m-\Delta:m}, h^{\beta(p)}_{m-\Delta:m} = h^{\alpha(q)}_{m-\Delta:m}, h^{\beta(q)}_{m-\Delta:m}, \tag{12}$$

then we *merge* them by eliminating diplotype q and giving its weight to diplotype p (after adjusting for the relative posterior probabilities of diplotypes p and q). Explicitly, we augment the weight, or "multiplicity," of diplotype p as follows:

$$w^{(p)} \leftarrow w^{(p)} + w^{(q)} \cdot \frac{\hat{P}(h^{\alpha(q)}_{1:m}, h^{\beta(q)}_{1:m} \mid g_{1:m})}{\hat{P}(h^{\alpha(p)}_{1:m}, h^{\beta(p)}_{1:m} \mid g_{1:m})}. \tag{13}$$

We rapidly identify diplotypes to be merged by maintaining bitmasks representing their alleles in the interval $[m-\Delta, m]$; identifying duplicate bitmask pairs thus amounts to sorting integer pairs.

Second, we sort all remaining diplotypes by weighted posterior probability $w^{(p)} \cdot \hat{P}(h^{\alpha(p)}_{1:m}, h^{\beta(p)}_{1:m} \mid g_{1:m})$ and *prune* the ensemble to the top P weighted diplotypes. Additionally, we prune any diplotypes with weighted probability less than ϵ^2 times that of the leading diplotype, where ϵ is the error probability parameter for single-SNP discrepancies (so ϵ^2 corresponds to the penalty of two errors).

Nature Genetics: doi:10.1038/ng.3679

Calling phase. For a given pair of consecutive heterozygous sites u, v, we identify the largest m such that the interval $[m-\Delta, m]$ contains u and v, and we estimate the relative phase of those sites by using the 2-dimensional marginal of the ensemble at the pair of positions (u, v):

$$P(h^{\alpha}_{u:v}, h^{\beta}_{u:v} \mid g_{1:m}) \propto\propto \sum_{p} w^{(p)} \cdot \hat{P}(h^{\alpha(p)}_{1:m}, h^{\beta(p)}_{1:m} \mid g_{1:m}) \cdot \mathbf{1}(h^{\alpha(p)}_{u:v}, h^{\beta(p)}_{u:v} = h^{\alpha}_{u:v}, h^{\beta}_{u:v}). \tag{14}$$

2.3.3 Refining phase via constrained searches

The procedure that we have described above works quite well, but we can improve speed via the following two-pass scheme. We first run a fast version of the propagation algorithm described above (with reduced computational parameters H=30, P=30, and Δ=10), obtaining phase calls and confidences at all pairs of consecutive hets and also obtaining error probabilities at homozygous genotypes (by marginalizing to single homozygous sites). We then identify phase calls and homozygous genotypes of confidence >99%, after which we perform a second, more accurate run of the propagation algorithm with increased computational parameters (H=100, P=50, and Δ=20) but much more limited branching: instead of considering all four haplotype extension combinations at each step, we consider only extensions consistent with the high-confidence inferences from the first step. Finally, we perform a constrained search with the same parameters in the reverse direction, and we combine the phase probabilities from the constrained forward and reverse searches (by averaging the marginal phase probabilities at consecutive hets) to produce final phase calls.

2.4 Optimizations

Having described the core theoretical components of the Eagle2 phasing algorithm, we now discuss a few key optimizations that achieve important constant-factor speedups in practice.

2.4.1 Condensing blocks of markers homozygous in the target sample

Thus far, we have framed the phase inference problem in terms of computing posterior probabilities $P(h^{\alpha}_{1:M}, h^{\beta}_{1:M} \mid g_{1:M})$, but we actually only wish to output inferred phase at markers m for which the target sample is heterozygous, i.e., $g_m = 1$. We note that the parental haplotypes $h^{\alpha}_{1:M}, h^{\beta}_{1:M}$ are not completely determined when $g_m \in \{0, 2\}$ because of the possibility of single-SNP errors. However, we have found that in practice, such errors are sufficiently strongly probabilistically disfavored that simply forcing copied haplotypes to agree with the target allele at homozygous sites results in almost no loss of accuracy (which seems reasonable given that alternative reference haplotypes without errors are usually available).

We make use of this observation to "horizontally" condense the haplotype space conditional on the genotypes of a given target sample. Specifically,

Nature Genetics: doi:10.1038/ng.3679

- we retain full haplotype information $h_{s(1:T)}$ at a subset of markers $\{s(1), \ldots, s(T)\} \subseteq \{1, \ldots, M\}$ containing all target-het sites (plus "spacer" sites that maintain a maximum distance of 0.5cM between consecutive sites $s(t), s(t+1)$ to avoid excessive coarsening of the model in runs of homozygosity), and
- within each interval $(s(t), s(t+1))$, we record only 1 bit e_t that indicates whether or not discrepancies exist between the haplotype segment $h_{s(t)+1:s(t+1)-1}$ and the (homozyous) target genotypes $g_{s(t)+1:s(t+1)-1}$.

This procedure produces a condensed representation $\{e_0 h_{s(1)} e_1 \cdots h_{s(T)} e_T\}$ of the reference haplotypes (Supplementary Fig. 1), as we describe in Section 2.4.2, but more importantly, it also enables a more efficient inference algorithm that skips forward one het at a time. In target-homozygous intervals $(s(t), s(t+1))$, we simply need to check the bit e_t: if $e_t = 1$, then we must end the haplotype segment currently being copied (and pay the transition cost of a recombination).

2.4.2 Creating individual-specific condensed haplotype trees

Performing inference on condensed haplotypes as described above substantially reduces the number of beam search propagation steps (from the number of markers M to roughly the number of hets in the target sample). However, because the condensed representation of the reference haplotypes is target sample-dependent, we need to rapidly generate a new HapHedge (on the condensed reference) for each target sample analyzed, and we also need each such HapHedge to be compact (because during multithreaded computation, each thread will need to build its own HapHedge for the target sample it is analyzing). Both of these goals are reasonable given the $O(MK)$-time and memory cost of building a HapHedge on K reference haplotypes, but a few optimizations help reduce the constant factors.

Representing condensed haplotypes in a HapHedge. As described above, a condensed haplotype $e_0 h_{s(1)} e_1 \cdots h_{s(T)} e_T$ directly corresponds to a bit string in which we alternately represent each actual allele $h_{s(t)}$ with 1 bit and each error bit e_t with 1 bit (Supplementary Fig. 1). Thus, the binary representation of a condensed haplotype requires $\approx$2 bits per target het.

Converting reference haplotypes into condensed bit string representations gives a smaller bit array; we next wish to build a "hedge" of haplotype prefix trees on this array rooted at every other bit (specifically, at the bits directly encoding alleles $h_{s(t)}$). We could directly build a HapHedge on the bit array, but one additional optimization saves time and memory. Given that we prohibit extension of a haplotype through any target-homozygous interval containing one or more discrepancies, we may truncate a prefix upon reaching an error bit $e_t = 1$. Upon truncating prefixes in this manner, many reference haplotypes collapse to the same prefix (for a given root location).

Nature Genetics: doi:10.1038/ng.3679

Consequently, we use a slightly different data structure (HapHedgeMulti, described in Section 3.4) that efficiently represents trees of haplotype prefixes with multiplicities.

Masking double-IBD regions. Previous authors have observed that some care is needed in scenarios where (i) the $2N$ reference haplotypes consist of imperfectly phased haplotype pairs from N individuals *and* (ii) the target sample contains "double-IBD" regions in which both chromosomal segments are shared identical-by-descent with a reference sample, such that the diploid genotypes of the reference and target samples match exactly in the region [12]. The first situation is guaranteed to arise when performing phasing without an external reference panel (because in this case, the "reference haplotypes" are simply current haplotype estimates for other samples). The second situation typically arises due to relatedness. Whatever the cause of this phenomenon, the result is that a model built on all reference haplotypes as-is will likely copy both double-IBD reference haplotypes when phasing the target sample, bringing along any phasing errors.

We apply the following procedure to guard against this pitfall. For a given target sample, we pre-process each pair of reference haplotypes (derived from a single individual) from left to right, identifying long regions of exact diploid genotype agreement. When such a region spans $>$20 split points $s(t)$, we mask both reference haplotypes throughout the double-IBD region (by setting error bits $e_t = 1$, which forces a switch upon entering the region). When such a region spans 10–20 split points $s(t)$, we mask the reference haplotype with the shorter distance since the last error.

Selecting *K* best reference haplotypes. All of the computation that we have described in the above paragraphs (to create a HapHedgeMulti data structure on a set of masked condensed reference haplotypes) scales linearly in the number of reference haplotypes and the number of markers. This computation is quite efficient, but as described in Section 2.2.3, when the number of reference haplotypes ($2N$) is very large, restricting to a subset of K individual-specific conditioning haplotypes saves time. To identify K "best" conditioning haplotypes, we rank reference haplotypes by computing the number of discrepancies between each reference haplotype and the homozygous genotypes of the target sample. As in our previous work [13], we perform computation on blocks of up to 64 SNPs at once using bit arithmetic; thus, the total computational cost of subset selection is $O(MN)$ with a very small constant factor (plus $O(N \log N)$ for sorting, which is negligible in practice).

We note that our discrepancy metric does not make use of inferred phase of the target genotypes and produces a single set of conditioning haplotypes to use for the entire region being phased; as such, our method for selecting conditioning haplotypes is much less sophisticated than that of SHAPEIT2 [12]. However, Eagle2 is able to condition on thousands of haplotypes (vs. K=100 by default for SHAPEIT2), which we suspect makes selection of conditioning haplotypes much less important. We also note that our $O(MN)$ cost for selection of conditioning haplotypes is

Nature Genetics: doi:10.1038/ng.3679

asymptotically slower than the method used by SHAPEIT3 (based on divisive clustering) [14]. Improving the scaling is a direction for future work (geared at million-sample data sets); for now, the constant factor is so small that this cost negligible for $N<$100K.

2.5 Imputing missing genotypes

The phasing algorithm we have described ignores genotype data at markers for which the target genotype is missing: such markers are ignored when condensing haplotypes. This behavior is ideal for the task of phasing a single target sample using a reference panel, but if we wish to perform phasing without a reference (or if we wish to use phased target samples to aid the phasing of other target samples), we also need to impute missing genotypes.

It turns out that we can harness the HapHedge and beam search machinery to impute missing genotypes during phasing at almost no additional computational cost. The key idea is that a haplotype prefix in a (condensed) HapHedge—which represents a cluster of haplotypes that match (at non-ignored sites) along the prefix—can be "lifted" back to a specific reference haplotype in the cluster. This fact is clear from our radix tree-based HapHedge implementation: we efficiently represent tree edges via reference haplotype indices. Thus, given a (condensed) prefix segment used in a diplotype, we can look up a reference haplotype that maps to this prefix, and then we can check which alleles the reference haplotype has at any ignored sites within the prefix.

More precisely, given a diplotype ensemble $\{(w^{(p)}, h^{\alpha(p)}_{1:s(t)}, h^{\beta(p)}_{1:s(t)})\}_p$, we impute missing target genotypes in the interval $(s(t-\Delta), s(t-\Delta+1))$—i.e., between split points $t-\Delta$ and $t-\Delta+1$—using the following sampling procedure (repeated 10 times). First, we sample a diplotype p from the ensemble (according to the weighted posterior probabilities we have computed). Then, for each of the haplotypes $h^{\alpha(p)}_{1:s(t)}$ and $h^{\beta(p)}_{1:s(t)}$ comprising the diplotype, we sample recombination splits from right to left along the haplotype according to recursion (10). This procedure splits each of the haplotypes $h^{\alpha(p)}_{1:s(t)}$ and $h^{\beta(p)}_{1:s(t)}$ into haplotype segments corresponding to HapHedge prefixes. Usually, no recombination splits the interval $(s(t-\Delta), s(t-\Delta+1))$, in which case we take the segment of $h^{\alpha(p)}_{1:s(t)}$ (resp. $h^{\beta(p)}_{1:s(t)}$) containing this interval, lift it to a reference haplotype, and have the reference haplotype vote on missing target alleles in $(s(t-\Delta), s(t-\Delta+1))$ according to its alleles at those sites. In the event that a recombination splits the interval $(s(t-\Delta), s(t-\Delta+1))$, we need to estimate where exactly the recombination took place (to know how far right to extend the left reference haplotype and how far left to extend the right reference haplotype). We do so by weighting each possible split point (between consecutive SNPs m and $m+1$ in the interval $(s(t-\Delta), s(t-\Delta+1))$) according to the genetic distance between m and $m+1$.

The above description captures the main thrust of our missing imputation method, but we mention one additional engineering subtlety. Although we have no absolute assignment of haplotypes to maternal vs. paternal chromosomes, we still need to align the pairs of votes we obtain across different samples from the above procedure. We do so by locally aligning each sampled diplotype

Nature Genetics: doi:10.1038/ng.3679

to the final phased haplotypes called by our phasing algorithm: when imputing missing genotypes in the interval $(s(t-\Delta), s(t-\Delta+1))$, we find the closest heterozygous split site and align the sampled diplotype to the final phased haplotypes according to their alleles at this site.

In theory, the procedure we have described could be extended to efficiently perform GWAS imputation (of untyped sites) during phasing. However, we have observed that it achieves slightly lower accuracy compared to standard imputation methods (e.g., Beagle v4.1 [25] and Minimac3). We suspect that the reason is that imputation and phasing are sensitive to different aspects of modeling: in particular, imputation requires careful modeling of both the maternal and paternal haplotypes at all locations, whereas correctly identifying one long matching haplotype is enough to perform accurate phasing. Thus, we currently believe GWAS imputation is still best done via the pre-phasing paradigm [27] of phasing followed by haploid imputation (on phased haplotypes). However, the algorithm above is well-suited to imputing of a small fraction of missing genotypes (typically a few percent) during within-cohort pre-phasing: in this setting, achieving optimal imputation accuracy at the missing sites is not neeeded to achieve the goal of high phasing accuracy.

3 HapHedge data structure

The HapHedge data structure represents a sequence of haplotype prefix trees (i.e., binary trees on haplotype prefixes) rooted at a given set of starting positions along a chromosome (Figure 1). The key features of the HapHedge are linear-time construction, linear-memory representation, and constant-time prefix extension (all with small constant factors). Specifically, the computational costs associated with a HapHedge on K haplotypes and M markers are as follows:

- $O(MK)$-time construction using the positional Burrows-Wheeler transform (PBWT) [20].
- $O(MK)$-memory representation of the full "hedge" using radix trees. More precisely, to represent M_t trees, $12M_tK + MK/8$ bytes are required. (If we begin a new tree every s positions, $M_t = M/s$.)
- $O(1)$-time extension of a haplotype prefix (with haplotype frequency lookup).

We first describe the in-memory representation of a HapHedge and show how it supports constant-time haplotype extension; we then describe HapHedge construction.

3.1 Compact representation of haplotype prefix trees in memory

The HapHedge data structure represents a sequence of haplotype prefix trees using *radix trees* backed by a *shared bit array* containing all input haplotypes. In general, a radix tree compactifies a prefix tree by merging any non-branching internal node with its parent. Each edge of a radix tree is thus labeled with a string (representing merged edges) rather than a single character.

Nature Genetics: doi:10.1038/ng.3679

In our case, we wish to represent many trees representing different suffixes of the same underlying set of K haplotype bit strings. As such, we can represent an edge label in constant space using only a pointer to the appropriate reference haplotype and the length of the label. We employ this strategy as follows.

- We store the reference haplotypes in a `bitArray` using MK bits.
- For each prefix start location, we store a radix tree using the following data layout:
  - 1 integer `hap0`: the index of the lexicographically first haplotype prefix
  - $K-1$ `nodes`, each containing:
    - 1 integer `mSplit`: the location of the next branch (i.e., the next location polymorphic among haplotype prefixes in the current subtree, or M if all haplotypes in the subtree are identical)
    - 1 integer `count0`: the number of haplotype prefixes in the 0-subtree at `mSplit`
    - 1 integer `hap1`: the index of the lexicographically first haplotype prefix in the 1-subtree at `mSplit`.

We note that the specification above does not contain any explicit pointers to indices of child nodes. We are able to avoid storing such pointers by storing the $K-1$ nodes of each radix tree in the order that they are encountered during a pre-order traversal of the tree. Assuming this ordering is chosen, the left child of a node is simply the node that follows it in memory, and the right child of a node is the node `count0` nodes after it. We present an example in Supplementary Fig. 2. (In the orientation of Supplementary Fig. 2, "left" and "right" children are depicted as upper and lower child nodes.)

We also note that if the data contains identical haplotypes (such that a radix tree in the usual sense would contain a leaf representing $k \geq 2$ identical haplotypes), we split any such leaves into subtrees with $k-1$ internal nodes. This way, the (extended) radix tree representing the haplotypes always has exactly $K-1$ internal nodes, and we preserve the property that the left child of a node is simply the node that follows it in memory and the right child of a node is the node `count0` nodes after it. The splitting of identical haplotypes into a subtree can be done arbitrarily; in our implementation, we linearly peel off one haplotype at a time.

3.2 Constant-time haplotype prefix extension

To follow a haplotype prefix along a radix tree represented in the above form, we need to keep track of some information beyond the current marker and the index of the current node. Specifically, the complete state information that we maintain during prefix lookup is:

- 1 integer `m`: the current marker

Nature Genetics: doi:10.1038/ng.3679

- 1 integer `node`: the index of the node at the end of the current edge
- 1 integer `hap`: the index of the lexicographically first haplotype prefix in the current subtree
- 1 integer `count`: the number of haplotype prefixes in the current subtree.

In particular, we note that the state for the root node of the radix tree starting at position `m0` is $\{$`m=m0`, `node=0`, `hap=hap0`, `count=`$K\}$. (We could of course have included the `hap` and `count` data fields in the nodes of the HapHedge, but doing so would increase its memory footprint.)

The logic for extending a haplotype prefix (with state information) to the `nextBit` (either 0 or 1) is probably easiest understood in pseudocode.

Pseudocode for extending a haplotype prefix.

```
INPUT:
- state.{m,node,hap,count}: current state information
- nextBit: bit (0 or 1) with which to attempt extension of prefix

OUTPUT:
- if successful extension, update state and return SUCCESS
- else, return FAILURE

function extend(state, nextBit) {
  if (state.count == 1 || nodes[state.node].mSplit > state.m) {
    if (bitArray(state.hap, state.m) != nextBit)
      return FAILURE
  }
  else {
    if (nextBit == 0) {
      state.count = nodes[state.node].count0
      state.node++
    }
    else {
      state.hap = nodes[state.node].hap1
      c0 = nodes[state.node].count0
      state.node += c0
      state.count -= c0
    }
  }
  state.m++
  return SUCCESS
}
```


Nature Genetics: doi:10.1038/ng.3679

3.3 Linear-time HapHedge construction

The key insight for efficient HapHedge construction is that the positional Burrows-Wheeler transform [20] already encodes all the information we need to construct the haplotype radix tree at each desired root marker in linear time. Specifically, Algorithm 2 of ref. [20] shows how to iteratively construct a *positional suffix array* `a[]` and *divergence array* `d[]` at each marker. (Ref. [20] works from left to right and hence constructs positional prefix arrays rather than suffix arrays, but we need only reverse the direction of processing.) At each marker m, the positional suffix array `a[]` precisely gives the lexicographic order on haplotype prefixes starting from m—i.e., the in-order traversal of leaves in the radix tree we wish to construct—and the divergence array `d[]` encodes the branching depths `mSplit` of internal nodes. Thus, if we wish to construct a radix tree on prefixes starting at m, we can perform an in-order traversal of the tree (before explicitly representing it) by walking through the `a[]` and `d[]` arrays, recovering the tree topology as we go based on relative divergence depths `d`.

The observations above give the following two-step algorithm for constructing a radix tree rooted at marker m (on reaching m while running PBWT Algorithm 2 from right to left). Given the matrices `a[]` and `d[]` for marker m, we first construct a temporary (less memory-efficient) explicit representation of the radix tree using the in-order traversal described above. Second, we perform a pre-order traversal of our temporary representation to generate the memory-efficient representation we desire. For details, see the implementation in `HapHedge.cpp`.

3.4 Extension to haplotypes with multiplicities: HapHedgeMulti

As we noted in Section 2.4.2, it is advantageous to modify our radix tree representation when representing condensed haplotypes because the number of unique condensed haplotype prefixes (after truncating at error bits $e_t = 1$) is typically much smaller than the number of input haplotypes. We call the modified data structure the HapHedgeMulti. Conceptually, the ideas behind the representation of haplotypes and the construction and prefix extension algorithms remain unchanged, so we only highlight a few main implementational differences.

- A HapHedgeMulti radix tree contains only $K_{\text{uniq}} - 1$ internal nodes (vs. $K - 1$ nodes in a HapHedge radix tree), where K_{uniq} is the number of unique prefixes.

- Each node of a HapHedgeMulti radix tree contains 5 integer data fields: the 3 fields `mSplit`, `count0`, and `hap1` from before, plus 2 new fields `node0` and `node1` giving the indices of child nodes (or -1 if a child is a leaf). (These additional fields are needed because node indices can no longer be inferred from counts due to haplotype multiplicities.)

- Truncation of prefixes at bits indicating errors at target homs (Section 2.4.2) is easily accomplished while running PBWT Algorithm 2 by changing values at the end of the divergence

Nature Genetics: doi:10.1038/ng.3679

array `d[]` to M. Specifically, if bit m encodes an error bit e_t, then after the first prefix with 1 at bit m, we change `d[*]` to M for all subsequent prefixes (which also must have 1 at bit m). This modification effectively equates all prefixes with 1 at bit m, thus accomplishing the truncation, and the equivalence is naturally propagated to radix tree construction and to subsequent PBWT iterations (both of which only see information about already-processed markers as represented by the matrices `a[]` and `d[]`.

Again, implementation details can be found in the file `HapHedge.cpp`.

4 Appendix: Recombination probability models

Here we provide a more complete discussion of equation (5) for the probability that a copied haplotype has length between u and v Morgans. This probability is equal to the integral (from u to v) of the IBD length distribution: that is, the distribution of lengths of most recent shared tracts between a given haplotype and a set of K reference haplotypes (in a population containing $2N_e$ haplotypes).

We first consider the Li-Stephens model [21]. Under this model, haplotype copying is memoryless (as one moves left-to-right across the genome), and the IBD length distribution decays exponentially with rate $4N_e/K$. Letting a denote the inverse of this rate (i.e., the mean IBD length), this distribution is given by

$$f_{\text{Li}-\text{Stephens}}(x) = \frac{1}{a} e^{-x/a}, \tag{15}$$

where x denotes IBD length.

In contrast, under the coalescent model, haplotype copying is not memoryless. Intuitively, the longer one has spent copying a haplotype, the longer one should expect to continue copying that haplotype, because long shared haplotypes indicate recent IBD. The full coalescent model is complex [36], but the SMC model [34], which assumes that IBD segments are delimited by recombination events, provides a good approximation for long segments [35]. Under this model, the IBD length distribution is given by equation (10) of ref. [35] and has the form

$$f(x) = \frac{2/a}{(1 + x/a)^3}, \tag{16}$$

where a is again the mean IBD length. Integrating this density from u to v gives equation (5):

$$\int_u^v \frac{2/a}{(1 + x/a)^3} dx = \frac{1}{(1 + u/a)^2} - \frac{1}{(1 + v/a)^2}. \tag{17}$$

In practice, we lower-bound this probability at 10^{-6} to increase robustness to recombination map errors.

Despite our intuition that the coalescent-based model should better-represent true haplotype sharing, Eagle2 achieved near-identical phasing accuracy using either recombination probability model (Supplementary Table 11). Moreover, phasing accuracy was quite insensitive to the expected IBD length parameter a across the range of lengths we tested (0.5cM–4cM). These observations lead us to hypothesize that the bulk of the inference comes down simply to whether or not a long shared reference haplotype exists; if such a haplotype does exist, we suspect that the precise probabilistic modeling does not make much difference.

Nature Genetics: doi:10.1038/ng.3679

References

1. Tewhey, R., Bansal, V., Torkamani, A., Topol, E. J. & Schork, N. J. The importance of phase information for human genomics. *Nature Reviews Genetics* **12**, 215–223 (2011).

2. Browning, S. R. & Browning, B. L. Haplotype phasing: existing methods and new developments. *Nature Reviews Genetics* **12**, 703–714 (2011).

3. Stephens, M., Smith, N. J. & Donnelly, P. A new statistical method for haplotype reconstruction from population data. *American Journal of Human Genetics* **68**, 978–989 (2001).

4. Halperin, E. & Eskin, E. Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics* **20**, 1842–1849 (2004).

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

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

7. Browning, S. R. & Browning, B. L. 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**, 1084–1097 (2007).

8. Kong, A. *et al.* Detection of sharing by descent, long-range phasing and haplotype imputation. *Nature Genetics* **40**, 1068–1075 (2008).

9. Browning, B. L. & Browning, S. R. A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *American Journal of Human Genetics* **84**, 210–223 (2009).

10. Delaneau, O., Marchini, J. & Zagury, J.-F. A linear complexity phasing method for thousands of genomes. *Nature Methods* **9**, 179–181 (2012).

11. Williams, A. L., Patterson, N., Glessner, J., Hakonarson, H. & Reich, D. Phasing of many thousands of genotyped samples. *American Journal of Human Genetics* **91**, 238–251 (2012).

12. Delaneau, O., Zagury, J.-F. & Marchini, J. Improved whole-chromosome phasing for disease and population genetic studies. *Nature Methods* **10**, 5–6 (2013).

13. Loh, P.-R., Palamara, P. F. & Price, A. L. Fast and accurate long-range phasing in a uk biobank cohort. *Nature Genetics* (2016).

14. O'Connell, J. *et al.* Haplotype estimation for biobank-scale data sets. *Nature Genetics* (2016).

Nature Genetics: doi:10.1038/ng.3679

15. Snyder, M. W., Adey, A., Kitzman, J. O. & Shendure, J. Haplotype-resolved genome sequencing: experimental methods and applications. *Nature Reviews Genetics* **16**, 344–358 (2015).

16. van de Geijn, B., McVicker, G., Gilad, Y. & Pritchard, J. K. WASP: allele-specific software for robust molecular quantitative trait locus discovery. *Nature Methods* **12**, 1061–1063 (2015).

17. Kumasaka, N., Knights, A. J. & Gaffney, D. J. Fine-mapping cellular QTLs with RASQUAL and ATAC-seq. *Nature Genetics* (2015).

18. Das, S. *et al.* Next generation genotype imputation service and methods. *Nature Genetics* (2016). In press.

19. McCarthy, S. *et al.* A reference panel of 64,976 haplotypes for genotype imputation. *Nature Genetics* (2016). In press.

20. Durbin, R. Efficient haplotype matching and storage using the positional Burrows–Wheeler transform (PBWT). *Bioinformatics* **30**, 1266–1272 (2014).

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

22. Sudlow, C. *et al.* UK Biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLOS Medicine* **12**, 1–10 (2015).

23. Kvale, M. N. *et al.* Genotyping informatics and quality control for 100,000 Subjects in the Genetic Epidemiology Research on Adult Health and Aging (GERA) Cohort. *Genetics* **200**, 1051–1060 (2015).

24. Banda, Y. *et al.* Characterizing race/ethnicity and genetic ancestry for 100,000 subjects in the Genetic Epidemiology Research on Adult Health and Aging (GERA) cohort. *Genetics* **200**, 1285–1295 (2015).

25. Browning, B. L. & Browning, S. R. Genotype imputation with millions of reference samples. *The American Journal of Human Genetics* **98**, 116–126 (2016).

26. 1000 Genomes Project Consortium *et al.* A global reference for human genetic variation. *Nature* **526**, 68–74 (2015).

27. Howie, B., Fuchsberger, C., Stephens, M., Marchini, J. & Abecasis, G. R. Fast and accurate genotype imputation in genome-wide association studies through pre-phasing. *Nature Genetics* **44**, 955–959 (2012).

28. He, D., Han, B. & Eskin, E. Hap-seq: an optimal algorithm for haplotype phasing with imputation using sequencing data. *Journal of Computational Biology* **20**, 80–92 (2013).

29. Delaneau, O., Howie, B., Cox, A. J., Zagury, J.-F. & Marchini, J. Haplotype estimation using sequencing reads. *American Journal of Human Genetics* **93**, 687–696 (2013).

30. Sharp, K., Kretzschmar, W., Delaneau, O. & Marchini, J. Phasing for medical sequencing using rare variants and large haplotype reference panels. *Bioinformatics* (2016).

31. Chang, C. C. *et al.* Second-generation PLINK: rising to the challenge of larger and richer datasets. *GigaScience* **4**, 1–16 (2015).

32. Browning, S. R. Multilocus association mapping using variable-length Markov chains. *American Journal of Human Genetics* **78**, 903–913 (2006).

33. Browning, B. L. & Browning, S. R. Efficient multilocus association testing for whole genome association studies using localized haplotype clustering. *Genetic Epidemiology* **31**, 365–375 (2007).

34. McVean, G. A. & Cardin, N. J. Approximating the coalescent with recombination. *Philosophical Transactions of the Royal Society of London B: Biological Sciences* **360**, 1387–1393 (2005).

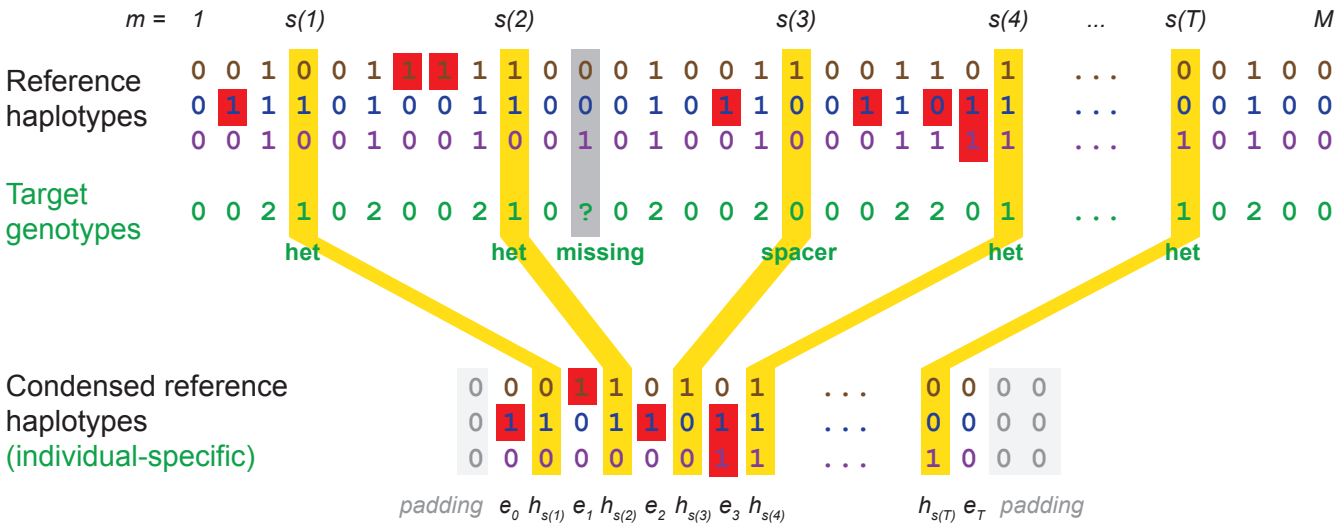
35. Palamara, P. F., Lencz, T., Darvasi, A. & Pe'er, I. Length distributions of identity by descent reveal fine-scale demographic history. *American Journal of Human Genetics* **91**, 809–822 (2012).

36. Harris, K. & Nielsen, R. Inferring demographic history from a spectrum of shared haplotype lengths. *PLOS Genetics* **9**, e1003521 (2013).

37. Drmanac, R. *et al.* Human genome sequencing using unchained base reads on self-assembling DNA nanoarrays. *Science* **327**, 78–81 (2010).

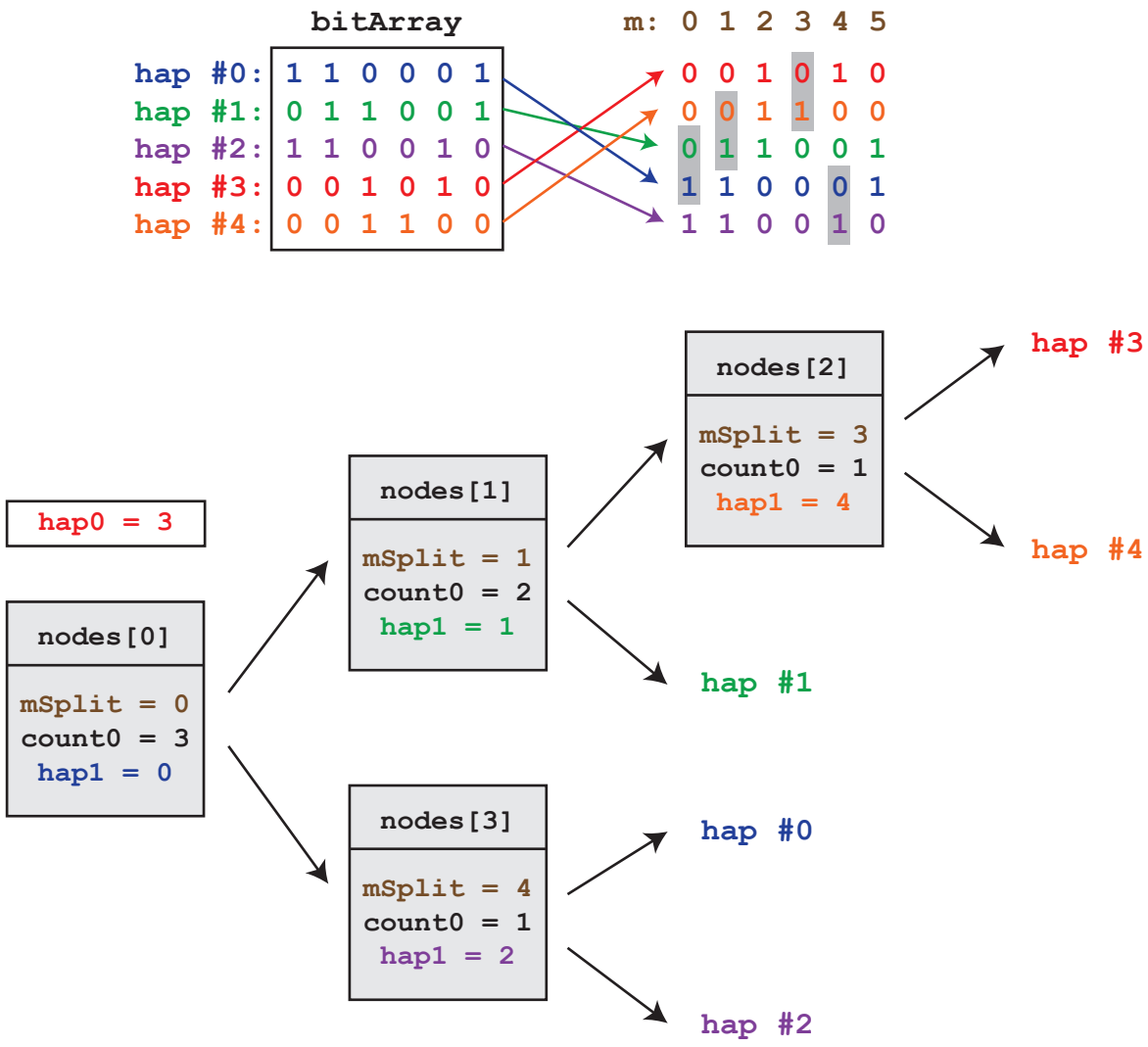
38. Huang, J. *et al.* Improved imputation of low-frequency and rare variants using the UK10K haplotype reference panel. *Nature Communications* **6** (2015).

Nature Genetics: doi:10.1038/ng.3679



Supplementary Figure 1. Conversion of reference haplotypes to condensed form. This figure illustrates the procedure used to create condensed reference haplotypes given the genotypes of a specific target sample (Supplementary Note Section 2.4.2). Alleles are directly coded in 1 bit each at a subset of sites $s(1), \ldots, s(T)$ consisting of target het sites plus spacer sites as needed to break up runs of homozygosity (ROH). In each region between directly-coded alleles, all target genotypes are either homozygous or missing; we encode the presence or absence of discrepancies between the reference haplotype and the target in 1 bit. (Finally, we pad each condensed haplotype with $h_{s(0)} = 0$ and $h_{s(T+1)} = 0, e_{T+1} = 0$ for implementational convenience.)

Nature Genetics: doi:10.1038/ng.3679



Supplementary Figure 2. A haplotype prefix tree in radix format. This example shows the HapHedge representation of a single tree on $K = 5$ haplotypes rooted at the first of $M = 6$ markers. A HapHedge will typically contain many trees rooted at different starting positions, all sharing the same `bitArray`, but each with its own `hap0` and `nodes`.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 1. Data sets analyzed.

Data set	Samples (N)	Indep. trios	Markers (M)	Heterozygosity	Missingness
UK Biobank	152,248	72	707,524	0.173	0.9%
GERA European chip	62,318	200	657,184	0.257	0.6%
GERA African chip	3,826	3	851,970	0.216	1.1%
GERA East Asian chip	5,188	7	694,877	0.240	0.6%
GERA Latino chip	7,154	3	776,817	0.225	1.4%

We report basic sample and SNP statistics for the UK Biobank and GERA data sets analyzed. The UK Biobank statistics reflect the light QC we performed (described in Online Methods); the GERA statistics correspond exactly to the data we obtained from dbGaP. "Independent trios" indicate numbers of non-overlapping mother-father-child trios. "Markers" indicate autosomal SNP counts. Heterozygosity and missingness are means over all samples and SNPs.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 2. Running time and accuracy of reference-based phasing in UK Biobank benchmarks.

(a) CPU time per target genome

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle1	1.2 min	1.4 min	1.6 min	2.0 min
Eagle2 *K*=5K	1.1 min	1.1 min	1.1 min	1.2 min
Eagle2 *K*=10K	1.5 min	1.6 min	1.6 min	1.6 min
Eagle2 *K*=20K	2.4 min	2.3 min	2.3 min	2.5 min
Eagle2 *K*=40K	3.4 min	4.7 min	4.2 min	4.4 min
SHAPEIT2 *K*=50 –no-mcmc	2.1 min	2.6 min	3.6 min	4.7 min
SHAPEIT2 *K*=100 –no-mcmc	2.6 min	3.1 min	3.7 min	5.3 min
SHAPEIT2 *K*=200 –no-mcmc	3.7 min	4.1 min	4.9 min	6.4 min
SHAPEIT2 *K*=400 –no-mcmc	5.6 min	6.1 min	6.9 min	8.5 min
SHAPEIT2 *K*=50	13.7 min	20.8 min	30.3 min	53.8 min
SHAPEIT2 *K*=100	18.0 min	26.0 min	36.0 min	60.8 min
SHAPEIT2 *K*=200	26.2 min	34.2 min	44.9 min	71.0 min
SHAPEIT2 *K*=400	41.6 min	49.7 min	60.9 min	87.8 min

(b) Mean switch error rate (s.e.m.)

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle1	1.35% (0.04%)	0.88% (0.03%)	0.65% (0.03%)	0.40% (0.02%)
Eagle2 *K*=5K	0.98% (0.03%)	0.75% (0.03%)	0.63% (0.03%)	0.46% (0.02%)
Eagle2 *K*=10K	0.87% (0.03%)	0.65% (0.02%)	0.53% (0.02%)	0.39% (0.02%)
Eagle2 *K*=20K	0.79% (0.02%)	0.58% (0.02%)	0.47% (0.02%)	0.34% (0.02%)
Eagle2 *K*=40K	0.76% (0.02%)	0.54% (0.02%)	0.43% (0.02%)	0.31% (0.02%)
SHAPEIT2 *K*=50 –no-mcmc	3.82% (0.06%)	3.64% (0.07%)	3.58% (0.08%)	3.64% (0.07%)
SHAPEIT2 *K*=100 –no-mcmc	2.35% (0.05%)	2.14% (0.05%)	2.09% (0.05%)	2.06% (0.06%)
SHAPEIT2 *K*=200 –no-mcmc	1.65% (0.04%)	1.39% (0.04%)	1.30% (0.04%)	1.23% (0.04%)
SHAPEIT2 *K*=400 –no-mcmc	1.28% (0.03%)	1.01% (0.03%)	0.89% (0.03%)	0.78% (0.03%)
SHAPEIT2 *K*=50	1.23% (0.03%)	0.86% (0.03%)	0.69% (0.03%)	0.49% (0.02%)
SHAPEIT2 *K*=100	1.04% (0.03%)	0.74% (0.03%)	0.58% (0.02%)	0.41% (0.02%)
SHAPEIT2 *K*=200	0.93% (0.03%)	0.65% (0.02%)	0.52% (0.02%)	0.37% (0.02%)
SHAPEIT2 *K*=400	0.85% (0.02%)	0.60% (0.02%)	0.47% (0.02%)	0.34% (0.02%)

(This table provides numeric data plotted in Figure 2 along with additional benchmarks.) We performed reference-based phasing using reference panels generated from N_{ref} = 15,000, 30,000, 50,000, or 100,000 UK Biobank samples. We analyzed chromosomes 1, 5, 10, 15, and 20 ($\approx$25% of the genome) in 4-threaded computation on 2.27 GHz Intel Xeon L5640 processors. To obtain CPU time per target genome, we then scaled up computation time by a factor of 4; see Supplementary Table 3 for details. We assessed accuracy on the 70 UK Biobank European-ancestry trio children, aggregating switch errors over the five chromosomes analyzed, and computing means and s.e.m. over the 70 samples. The default values of K for SHAPEIT2 and Eagle2 are 100 and 10,000, respectively.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 3. Computational cost of reference-based phasing in UK Biobank benchmarks at different target sample sizes.

(a) CPU time and memory for phasing chr1,5,10,15,20 of N_{target} = 72 samples

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle1	0.6 hr / 2.8 GB	0.9 hr / 4.8 GB	1.2 hr / 6.7 GB	2.0 hr / 10.9 GB
Eagle2 *K*=5K	0.4 hr / 1.4 GB	0.4 hr / 2.0 GB	0.4 hr / 2.9 GB	0.6 hr / 5.5 GB
Eagle2 *K*=10K	0.5 hr / 1.4 GB	0.5 hr / 2.1 GB	0.6 hr / 3.0 GB	0.7 hr / 5.4 GB
Eagle2 *K*=20K	0.7 hr / 1.6 GB	0.8 hr / 2.3 GB	0.8 hr / 3.4 GB	0.9 hr / 5.7 GB
Eagle2 *K*=40K	1.0 hr / 1.9 GB	1.4 hr / 2.5 GB	1.3 hr / 3.8 GB	1.5 hr / 6.2 GB
SHAPEIT2 *K*=50 –no-mcmc	1.1 hr / 0.6 GB	2.1 hr / 1.0 GB	3.0 hr / 1.7 GB	5.8 hr / 3.3 GB
SHAPEIT2 *K*=100 –no-mcmc	1.4 hr / 0.7 GB	2.3 hr / 1.1 GB	3.4 hr / 1.8 GB	5.9 hr / 3.4 GB
SHAPEIT2 *K*=200 –no-mcmc	1.7 hr / 0.8 GB	2.7 hr / 1.2 GB	3.7 hr / 2.0 GB	6.1 hr / 3.5 GB
SHAPEIT2 *K*=400 –no-mcmc	2.3 hr / 1.2 GB	3.3 hr / 1.6 GB	4.4 hr / 2.3 GB	7.0 hr / 3.8 GB
SHAPEIT2 *K*=50	4.3 hr / 0.6 GB	6.9 hr / 1.0 GB	10.7 hr / 1.7 GB	19.2 hr / 3.3 GB
SHAPEIT2 *K*=100	5.4 hr / 0.6 GB	8.4 hr / 1.0 GB	12.2 hr / 1.7 GB	20.7 hr / 3.3 GB
SHAPEIT2 *K*=200	7.8 hr / 0.6 GB	10.7 hr / 1.0 GB	14.5 hr / 1.7 GB	23.4 hr / 3.3 GB
SHAPEIT2 *K*=400	12.4 hr / 0.6 GB	15.4 hr / 1.0 GB	19.4 hr / 1.7 GB	28.5 hr / 3.3 GB

(b) CPU time and memory for phasing chr1,5,10,15,20 of N_{target} = 1,072 samples

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle1	5.7 hr / 3.0 GB	6.7 hr / 4.7 GB	7.8 hr / 6.8 GB	10.3 hr / 11.2 GB
Eagle2 *K*=5K	5.0 hr / 1.4 GB	5.1 hr / 2.1 GB	5.1 hr / 3.1 GB	5.5 hr / 5.4 GB
Eagle2 *K*=10K	6.9 hr / 1.5 GB	7.1 hr / 2.2 GB	7.3 hr / 3.1 GB	7.3 hr / 5.6 GB
Eagle2 *K*=20K	10.6 hr / 1.7 GB	10.6 hr / 2.4 GB	10.6 hr / 3.4 GB	11.4 hr / 5.7 GB
Eagle2 *K*=40K	15.2 hr / 2.0 GB	21.0 hr / 2.9 GB	18.7 hr / 3.8 GB	20.0 hr / 6.1 GB
SHAPEIT2 *K*=50 –no-mcmc	9.8 hr / 1.6 GB	12.9 hr / 2.0 GB	18.1 hr / 2.7 GB	25.2 hr / 4.3 GB
SHAPEIT2 *K*=100 –no-mcmc	12.2 hr / 1.7 GB	15.3 hr / 2.1 GB	18.7 hr / 2.8 GB	28.0 hr / 4.4 GB
SHAPEIT2 *K*=200 –no-mcmc	16.9 hr / 1.9 GB	19.8 hr / 2.3 GB	23.9 hr / 3.0 GB	32.8 hr / 4.5 GB
SHAPEIT2 *K*=400 –no-mcmc	25.6 hr / 2.2 GB	28.8 hr / 2.6 GB	33.0 hr / 3.3 GB	42.5 hr / 4.8 GB
SHAPEIT2 *K*=50	61.6 hr / 1.6 GB	93.5 hr / 2.0 GB	137.1 hr / 2.7 GB	243.4 hr / 4.2 GB
SHAPEIT2 *K*=100	80.3 hr / 1.6 GB	116.7 hr / 2.0 GB	162.3 hr / 2.7 GB	274.0 hr / 4.2 GB
SHAPEIT2 *K*=200	117.1 hr / 1.6 GB	153.1 hr / 2.0 GB	201.7 hr / 2.7 GB	319.1 hr / 4.2 GB
SHAPEIT2 *K*=400	185.5 hr / 1.6 GB	222.4 hr / 2.0 GB	273.0 hr / 2.7 GB	394.2 hr / 4.2 GB

To benchmark per-sample computational cost of reference-based phasing, we performed two sets of analyses: one in which we phased only N_{target} = 72 UK Biobank trio children, and another in which we phased an additional 1,000 randomly selected samples. We then subtracted the N_{target} = 72 CPU times from the N_{target} = 1,072 CPU times to obtain the incremental cost of phasing 1,000 samples. This procedure was necessary to adjust for initialization costs (e.g., reading the reference data and synchronizing it with the target data), which account for a non-neglibible fraction of total computational cost when N_{target} is small. Finally, we divided by 1,000 to obtain per-sample costs and multiplied by 4 to scale up from 25% of the genome (chr1,5,10,15,20) to obtain the numbers plotted in Fig. 2a and reported in Supplementary Table 2.

We performed runs using 4 cores of a 2.27 GHz Intel Xeon L5640 processor. We chose to report CPU times rather than wall clock run times because we observed less run-to-run variability in CPU times on the shared compute cluster we used for these benchmarks. (Both Eagle and SHAPEIT perform efficient multithreading, so the choice of CPU time vs. wall time has little effect on relative performance.)

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 4. Accuracy of reference-based phasing in GERA benchmarks using varying numbers of conditioning haplotypes.

(a) Mean switch error rate (s.e.m.), trio parents

Method	European chip N_{ref} = 61,684 N_{target} = 400	African chip N_{ref} =3,817 N_{target} = 6	East Asian chip N_{ref} = 5,164 N_{target} = 14	Latino chip N_{ref} = 7,144 N_{target} = 6
SHAPEIT2 *K*=100	1.31% (0.04%)	3.21% (0.08%)	2.24% (0.12%)	2.42% (0.05%)
SHAPEIT2 *K*=200	1.23% (0.04%)	2.93% (0.07%)	2.11% (0.11%)	2.29% (0.05%)
SHAPEIT2 *K*=400	1.17% (0.04%)	2.75% (0.06%)	2.02% (0.11%)	2.18% (0.04%)
SHAPEIT2 *K*=100 –no-mcmc	2.83% (0.05%)	5.29% (0.15%)	3.34% (0.14%)	3.91% (0.08%)
SHAPEIT2 *K*=200 –no-mcmc	2.11% (0.05%)	4.16% (0.11%)	2.80% (0.13%)	3.14% (0.06%)
SHAPEIT2 *K*=400 –no-mcmc	1.72% (0.05%)	3.51% (0.09%)	2.48% (0.12%)	2.72% (0.06%)
Eagle1	1.36% (0.04%)	3.87% (0.09%)	2.71% (0.14%)	2.65% (0.05%)
Eagle2 *K*=5K	1.32% (0.04%)	2.64% (0.06%)	2.00% (0.10%)	2.17% (0.04%)
Eagle2 *K*=10K	1.24% (0.04%)	2.48% (0.05%)	1.93% (0.10%)	2.08% (0.04%)
Eagle2 *K*=20K	1.17% (0.04%)	2.48% (0.05%)	1.95% (0.10%)	2.06% (0.04%)

(b) Mean switch error rate (s.e.m.), trio children

Method	European N_{ref} = 61,684 N_{target} = 200
SHAPEIT2 *K*=100	0.90% (0.04%)
SHAPEIT2 *K*=200	0.82% (0.04%)
SHAPEIT2 *K*=400	0.75% (0.04%)
SHAPEIT2 *K*=100 –no-mcmc	2.46% (0.06%)
SHAPEIT2 *K*=200 –no-mcmc	1.73% (0.05%)
SHAPEIT2 *K*=400 –no-mcmc	1.31% (0.05%)
Eagle1	0.95% (0.04%)
Eagle2 *K*=5K	0.93% (0.04%)
Eagle2 *K*=10K	0.84% (0.04%)
Eagle2 *K*=20K	0.78% (0.04%)

(This table provides numeric data plotted in Figure 3 along with additional benchmarks.)

(a) We phased trio parents in each GERA sub-cohort using a reference panel generated from all other non-familial samples in the same sub-cohort. We ran each method with default parameter settings on all 22 autosomes and computed aggregate mean switch error rates (s.e.m.). Standard errors for the European-ancestry sub-cohort are over 400 parent samples. Standard errors for the other three sub-cohorts are over 25 SNP blocks.

(b) We phased GERA trio children on the European chip sub-cohort using a reference panel generated from all other non-familial European chip samples. (We did not perform this analysis on the other sub-cohorts due to the low numbers of trios.) We note that accuracy is higher across all methods for trio children vs. trio parents, perhaps due to greater levels of (within-Europe) admixture in the children.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 5. Accuracy of reference-based phasing using the 1000 Genomes and HRC panels.

(a) Mean switch error rates (s.e.m.) using the 1000 Genomes Project Phase 3 panel

	CEU	CHS	PEL	PJL	YRI
Method	$N_{\text{target}} = 32$	$N_{\text{target}} = 31$	$N_{\text{target}} = 30$	$N_{\text{target}} = 15$	$N_{\text{target}} = 19$
SHAPEIT2	3.52% (0.06%)	6.51% (0.11%)	4.45% (0.16%)	4.75% (0.24%)	3.71% (0.11%)
SHAPEIT2 –no-mcmc	4.83% (0.05%)	7.79% (0.11%)	5.71% (0.17%)	6.41% (0.26%)	4.97% (0.11%)
Eagle1	5.40% (0.08%)	8.98% (0.14%)	6.57% (0.21%)	6.89% (0.35%)	6.72% (0.11%)
Eagle2	3.27% (0.06%)	6.38% (0.11%)	4.51% (0.19%)	4.35% (0.21%)	3.61% (0.10%)

(b) Mean switch error rates (s.e.m.) using the HRC panel

	CEU	CHS	PEL	PJL	YRI
Method	$N_{\text{target}} = 32$	$N_{\text{target}} = 31$	$N_{\text{target}} = 30$	$N_{\text{target}} = 15$	$N_{\text{target}} = 19$
SHAPEIT2	1.60% (0.05%)	7.78% (0.08%)	4.65% (0.15%)	5.97% (0.19%)	4.68% (0.10%)
SHAPEIT2 –no-mcmc	3.39% (0.08%)	9.56% (0.09%)	6.92% (0.10%)	9.00% (0.13%)	6.51% (0.09%)
Eagle1	1.89% (0.06%)	10.04% (0.10%)	6.30% (0.21%)	7.35% (0.17%)	7.76% (0.15%)
Eagle2	1.36% (0.04%)	7.52% (0.09%)	4.78% (0.16%)	5.22% (0.15%)	4.75% (0.09%)

We phased 1000 Genomes Phase 3 trio children on chromosome 1 using the 1000 Genomes Project Phase 3 panel or the Haplotype Reference Consortium panel (excluding these trios). To simulate a typical use case, we restricted the data to 31,853 markers (genotyped by 23andMe). We report mean switch error rates (s.e.m.) over trio samples in each population with >1 trio.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 6. Accuracy of phasing non-European UK Biobank samples using increasing numbers of European reference samples.

N_{matched}	Method	$N_{\text{mismatched}}$ 0	5,000	15,000	50,000	147,000
1,000	Eagle2	5.25% (0.21%)	5.23% (0.25%)	5.17% (0.24%)	4.95% (0.21%)	4.69% (0.22%)
	SHAPEIT2	5.89% (0.21%)	5.67% (0.23%)	5.87% (0.24%)	6.01% (0.24%)	5.83% (0.23%)
2,000	Eagle2	4.12% (0.17%)	4.17% (0.19%)	4.11% (0.20%)	4.06% (0.18%)	3.93% (0.17%)
	SHAPEIT2	4.82% (0.17%)	4.60% (0.22%)	4.80% (0.22%)	4.90% (0.22%)	4.88% (0.21%)
4,000	Eagle2	3.37% (0.16%)	3.26% (0.16%)	3.28% (0.16%)	3.29% (0.15%)	3.24% (0.15%)
	SHAPEIT2	3.96% (0.16%)	3.75% (0.16%)	3.83% (0.17%)	3.88% (0.18%)	4.00% (0.17%)

To investigate the effect of including ancestry-mismatched reference samples on reference-based phasing accuracy, we phased trio parents in the two non-European (specifically, Indian and Caribbean) UK Biobank trios using subsets of UK Biobank haplotypes (from a run of Eagle1 on all samples together) containing increasing numbers of European haplotypes. Specifically, we partitioned the non-trio samples into an "ancestry-matched" set of 4,708 samples that self-reported Mixed, Asian, or Black ancestry and an "ancestry-mismatched" set of the remaining 147,324 predominantly European samples. We then created reference panels containing combinations of 1,000, 2,000, or 4,000 ancestry-matched samples and 0, 5,000, 15,000, 50,000, or 147,000 ancestry-mismatched samples. We used these reference panels to phase the 4 non-European parents using either Eagle2 or SHAPEIT2. We report mean switch error rates over chromosomes 15–22 (s.e.m.).

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 7. Computational cost and accuracy of non-reference-based phasing in the UK Biobank cohort.

(a) Wall clock time and memory cost per genome

Method	N = 5K	N = 15K	N = 50K	N = 150K
SHAPEIT2	55.7 hr / 5.3 GB	230.9 hr / 15.2 GB	–	–
Eagle1	8.1 hr / 4.4 GB	30.4 hr / 8.0 GB	138.6 hr / 21.4 GB	715.7 hr / 56.7 GB
Eagle2 *K*=5K	10.1 hr / 4.8 GB	29.0 hr / 7.4 GB	98.0 hr / 17.1 GB	415.4 hr / 45.8 GB
Eagle2 *K*=10K	11.8 hr / 5.5 GB	34.2 hr / 7.4 GB	118.2 hr / 17.1 GB	464.3 hr / 45.8 GB
Eagle2 *K*=20K	12.9 hr / 5.4 GB	44.9 hr / 7.4 GB	153.2 hr / 17.2 GB	574.9 hr / 45.8 GB
Eagle2 *K*=40K	13.4 hr / 5.2 GB	59.9 hr / 8.1 GB	237.0 hr / 17.1 GB	806.5 hr / 45.8 GB

(b) Mean switch error rate (s.e.m.)

Method	N = 5K	N = 15K	N = 50K	N = 150K
SHAPEIT2	1.38% (0.03%)	0.86% (0.03%)	–	–
Eagle1	2.38% (0.05%)	1.30% (0.04%)	0.59% (0.03%)	0.31% (0.02%)
Eagle2 *K*=5K	1.36% (0.03%)	1.00% (0.03%)	0.65% (0.03%)	0.41% (0.02%)
Eagle2 *K*=10K	1.23% (0.03%)	0.85% (0.03%)	0.53% (0.02%)	0.35% (0.02%)
Eagle2 *K*=20K	1.23% (0.03%)	0.75% (0.02%)	0.46% (0.02%)	0.30% (0.02%)
Eagle2 *K*=40K	1.23% (0.03%)	0.73% (0.02%)	0.41% (0.02%)	0.27% (0.02%)

(This table provides numeric data plotted in Figure 5 along with benchmarks for additional parameter settings of Eagle2.) We benchmarked Eagle2 and other available phasing methods on N=5,000, 15,000 50,000, and 150,000 UK Biobank samples (including trio children and excluding trio parents). (**a**) Total wall clock time and maximum memory used for genome-wide phasing on a 16-core 2.60 GHz Intel Xeon E5-2650 v2 processor. (We analyzed a total of 174,595 markers on chromosomes 1, 5, 10, 15, and 20, representing ≈25% of the genome, and scaled up running times by a factor of 4; see Supplementary Table 8 for per-chromosome data.) SHAPEIT2 was unable to complete the N=50,000 chr1 and chr5 analyses and was unable to complete any of the N=150,000 analyses in 5 days, the run time limit for single compute jobs. (**b**) Mean switch error rate (s.e.m.) over 70 European-ancestry trios.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 8. Per-chromosome computational cost and accuracy of non-reference-based phasing in the UK Biobank cohort.

(a) Running time and memory cost

		SHAPEIT2	Eagle1	Eagle2			
N	chrom			K=5K	K=10K	K=20K	K=40K
5K	chr20	1.5 hr / 1.8 GB	0.2 hr / 1.3 GB	0.3 hr / 1.5 GB	0.3 hr / 2.2 GB	0.4 hr / 2.0 GB	0.3 hr / 2.3 GB
5K	chr15	1.8 hr / 2.1 GB	0.3 hr / 1.7 GB	0.4 hr / 2.3 GB	0.4 hr / 2.5 GB	0.5 hr / 2.3 GB	0.4 hr / 2.6 GB
5K	chr10	2.8 hr / 3.2 GB	0.4 hr / 2.4 GB	0.5 hr / 3.3 GB	0.6 hr / 3.5 GB	0.7 hr / 2.7 GB	0.7 hr / 3.3 GB
5K	chr5	3.4 hr / 3.9 GB	0.4 hr / 3.0 GB	0.6 hr / 3.0 GB	0.7 hr / 3.3 GB	0.7 hr / 4.1 GB	0.8 hr / 3.9 GB
5K	chr1	4.5 hr / 5.3 GB	0.7 hr / 4.4 GB	0.8 hr / 4.8 GB	0.9 hr / 5.5 GB	1.0 hr / 5.4 GB	1.1 hr / 5.2 GB
15K	chr20	5.9 hr / 5.1 GB	0.8 hr / 2.9 GB	0.8 hr / 2.8 GB	0.9 hr / 2.9 GB	1.3 hr / 3.3 GB	1.6 hr / 3.5 GB
15K	chr15	7.4 hr / 6.1 GB	1.0 hr / 3.4 GB	0.9 hr / 3.2 GB	1.1 hr / 3.4 GB	1.5 hr / 3.8 GB	2.0 hr / 4.1 GB
15K	chr10	10.8 hr / 9.5 GB	1.5 hr / 5.1 GB	1.6 hr / 4.5 GB	1.8 hr / 4.6 GB	2.2 hr / 5.2 GB	3.0 hr / 5.1 GB
15K	chr5	13.8 hr / 11.3 GB	1.7 hr / 6.1 GB	1.7 hr / 5.3 GB	2.0 hr / 5.3 GB	2.7 hr / 5.5 GB	3.6 hr / 6.1 GB
15K	chr1	19.8 hr / 15.2 GB	2.6 hr / 8.0 GB	2.3 hr / 7.4 GB	2.7 hr / 7.4 GB	3.6 hr / 7.4 GB	4.8 hr / 8.1 GB
50K	chr20	46.5 hr / 16.8 GB	3.9 hr / 7.7 GB	2.7 hr / 5.8 GB	3.1 hr / 5.7 GB	4.7 hr / 5.7 GB	6.2 hr / 5.7 GB
50K	chr15	59.0 hr / 20.2 GB	4.6 hr / 9.0 GB	3.3 hr / 7.0 GB	4.0 hr / 6.9 GB	5.0 hr / 7.0 GB	7.6 hr / 7.0 GB
50K	chr10	93.8 hr / 31.4 GB	6.6 hr / 12.9 GB	5.0 hr / 10.3 GB	6.4 hr / 10.3 GB	7.4 hr / 10.3 GB	13.2 hr / 10.3 GB
50K	chr5	–	8.2 hr / 15.4 GB	5.7 hr / 12.3 GB	6.8 hr / 12.3 GB	9.0 hr / 12.3 GB	13.6 hr / 12.3 GB
50K	chr1	–	11.4 hr / 21.4 GB	7.9 hr / 17.1 GB	9.2 hr / 17.1 GB	12.2 hr / 17.2 GB	18.6 hr / 17.1 GB
150K	chr20	–	19.9 hr / 20.1 GB	10.9 hr / 15.0 GB	12.3 hr / 15.0 GB	15.4 hr / 15.0 GB	21.7 hr / 15.0 GB
150K	chr15	–	24.4 hr / 24.1 GB	13.5 hr / 18.4 GB	15.2 hr / 18.3 GB	18.9 hr / 18.3 GB	26.6 hr / 18.3 GB
150K	chr10	–	35.2 hr / 34.2 GB	21.0 hr / 27.6 GB	22.9 hr / 27.6 GB	28.6 hr / 27.7 GB	39.9 hr / 27.7 GB
150K	chr5	–	41.3 hr / 40.5 GB	24.3 hr / 32.9 GB	27.6 hr / 32.9 GB	34.0 hr / 32.9 GB	48.5 hr / 33.0 GB
150K	chr1	–	58.1 hr / 56.7 GB	34.0 hr / 45.8 GB	38.1 hr / 45.8 GB	46.9 hr / 45.8 GB	65.0 hr / 45.8 GB

(b) Mean switch error rate (s.e.m.)

		SHAPEIT2	Eagle1	Eagle2			
N	chrom			K=5K	K=10K	K=20K	K=40K
5K	chr20	1.73% (0.06%)	2.86% (0.09%)	1.57% (0.05%)	1.46% (0.05%)	1.45% (0.05%)	1.45% (0.05%)
5K	chr15	1.66% (0.06%)	2.81% (0.09%)	1.58% (0.05%)	1.47% (0.05%)	1.49% (0.05%)	1.49% (0.05%)
5K	chr10	1.33% (0.04%)	2.29% (0.07%)	1.32% (0.04%)	1.20% (0.04%)	1.20% (0.04%)	1.20% (0.04%)
5K	chr5	1.21% (0.04%)	2.18% (0.07%)	1.25% (0.04%)	1.12% (0.04%)	1.11% (0.04%)	1.11% (0.04%)
5K	chr1	1.31% (0.04%)	2.26% (0.06%)	1.31% (0.04%)	1.15% (0.03%)	1.16% (0.03%)	1.16% (0.03%)
15K	chr20	1.05% (0.05%)	1.54% (0.06%)	1.14% (0.05%)	0.99% (0.05%)	0.88% (0.04%)	0.86% (0.04%)
15K	chr15	1.04% (0.05%)	1.52% (0.07%)	1.14% (0.05%)	1.03% (0.04%)	0.89% (0.04%)	0.86% (0.04%)
15K	chr10	0.82% (0.03%)	1.28% (0.05%)	0.95% (0.04%)	0.81% (0.03%)	0.75% (0.03%)	0.72% (0.03%)
15K	chr5	0.79% (0.03%)	1.18% (0.05%)	0.91% (0.04%)	0.77% (0.03%)	0.68% (0.03%)	0.66% (0.03%)
15K	chr1	0.81% (0.03%)	1.24% (0.04%)	0.98% (0.03%)	0.81% (0.03%)	0.71% (0.03%)	0.69% (0.03%)
50K	chr20	0.53% (0.04%)	0.67% (0.05%)	0.70% (0.05%)	0.57% (0.04%)	0.51% (0.04%)	0.46% (0.03%)
50K	chr15	0.57% (0.03%)	0.70% (0.04%)	0.72% (0.04%)	0.60% (0.03%)	0.53% (0.03%)	0.49% (0.03%)
50K	chr10	0.47% (0.02%)	0.59% (0.03%)	0.60% (0.03%)	0.50% (0.03%)	0.44% (0.03%)	0.40% (0.03%)
50K	chr5	–	0.53% (0.03%)	0.58% (0.03%)	0.49% (0.03%)	0.42% (0.02%)	0.37% (0.02%)
50K	chr1	–	0.55% (0.03%)	0.67% (0.03%)	0.55% (0.03%)	0.47% (0.03%)	0.40% (0.02%)
150K	chr20	–	0.32% (0.03%)	0.38% (0.03%)	0.33% (0.03%)	0.29% (0.03%)	0.27% (0.02%)
150K	chr15	–	0.36% (0.03%)	0.46% (0.03%)	0.37% (0.03%)	0.34% (0.03%)	0.31% (0.03%)
150K	chr10	–	0.32% (0.02%)	0.39% (0.03%)	0.34% (0.02%)	0.29% (0.02%)	0.26% (0.02%)
150K	chr5	–	0.29% (0.02%)	0.40% (0.03%)	0.33% (0.02%)	0.28% (0.02%)	0.25% (0.02%)
150K	chr1	–	0.29% (0.02%)	0.44% (0.03%)	0.36% (0.03%)	0.30% (0.02%)	0.27% (0.02%)

This table provides per-chromosome breakdowns of computational cost and phasing accuracy plotted in Figure 5 along with benchmarks for additional parameter settings of Eagle2. See the captions of Figure 5 and Supplementary Table 7 for benchmarking details.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 9. Accuracy of non-reference-based phasing in the GERA sub-cohorts.

Method	European chip N=62,084	African chip N=3,823	East Asian chip N=5,178	Latino chip N=7,150
SHAPEIT2 K=100	–	2.85% (0.07%)	2.02% (0.11%)	2.23% (0.05%)
Eagle1	1.34% (0.05%)	4.07% (0.10%)	3.09% (0.16%)	2.76% (0.06%)
Eagle2 K=5K	1.36% (0.04%)	2.81% (0.06%)	2.07% (0.11%)	2.20% (0.04%)
Eagle2 K=10K	1.25% (0.04%)	2.52% (0.05%)	1.97% (0.11%)	2.08% (0.04%)
Eagle2 K=20K	1.18% (0.04%)	2.50% (0.04%)	1.98% (0.11%)	2.06% (0.04%)

We benchmarked Eagle2 and other available phasing methods on the four GERA sub-cohorts (excluding all non-trio-parent samples from each trio family). For the European chip sub-cohort, we analyzed a total of 161,232 markers on chromosomes 1, 5, 10, 15, and 20; for the other sub-cohorts, we analyzed all 22 autosomes. Standard errors for the European-ancestry sub-cohort are over 400 parent samples. Standard errors for the other three sub-cohorts are over 25 SNP blocks. We did not benchmark SHAPEIT2 on the European-ancestry sub-cohort, as it would have required more than two weeks of computation.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 10. Phasing performance on sequence data.

Method	Mean switch error rate				Total CPU time per sample
	chr1	chr10	chr20	Combined (s.e.m.)	
SHAPEIT2	0.99%	0.90%	0.98%	0.96% (0.03%)	60.5 min
Eagle2 `--histFactor=1`	0.84%	0.77%	0.83%	0.82% (0.02%)	3.6 min
Eagle2 `--histFactor=2`	0.82%	0.75%	0.81%	0.80% (0.02%)	4.0 min
Eagle2 `--histFactor=4`	0.82%	0.76%	0.80%	0.80% (0.02%)	4.2 min

We benchmarked Eagle2 and SHAPEIT2 on sequence data from two independent trios (NA12877, NA12889, NA12890 and NA12878, NA12891, NA12892) belonging to the 17-member CEPH pedigree 1463. This data was part of the public 69-genome data set sequenced at 51–89x coverage by Complete Genomics and assembled using CGA Tools v1.6 and Analysis Pipeline v2.0.0 (ref. [37]). We phased chromosomes 1, 10, and 20 of the four trio parents using the merged 1000 Genomes Phase 3 + UK10K haplotype reference panel (EGAD00001000776; ref. [38]). After removing 3 individuals in the reference panel who belonged to the pedigree, the panel contained 6,282 samples. As in our previous benchmarks, we assessed switch errors according to gold standard trio phase, and we measured running time per sample not including constant initialization time required to load the data.

We ran SHAPEIT2 with `--window`=0.5 as recommended for sequence data [29]. We ran Eagle2 with its history length parameter set to 100 target hets (the default for analyzing genotype array data), and we also tested the effect of doubling or quadrupling the history length (thus considering haplotype matches spanning up to 200 or 400 target hets; Supplementary Note Sec. 2.2.1). We observed that Eagle2 achieved gains in accuracy and speed over SHAPEIT2 similar to the gains we observed in our benchmarks on genotype array data. Surprisingly, increasing Eagle2's history parameter had little effect on accuracy in these benchmarks. One possible explanation of this observation is that rare variants provide enough information about haplotype structure for there to be little gain in using extended haplotype information. Another possibility is that genotyping and/or phasing errors in the reference panel break up haplotypes at the length scale of 100 hets. Further work will be required to investigate whether this behavior occurs in other sequence data sets.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 11. Sensitivity of Eagle2 to recombination probability model.

(a) Mean switch error rate (s.e.m.)

Coalescent-based	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--expectIBDcM=0.5`	0.90% (0.03%)	0.67% (0.02%)	0.55% (0.02%)	0.40% (0.02%)
Eagle2 `--expectIBDcM=1`	0.87% (0.03%)	0.65% (0.02%)	0.53% (0.02%)	0.39% (0.02%)
Eagle2 `--expectIBDcM=2` (default)	0.87% (0.03%)	0.65% (0.02%)	0.53% (0.02%)	0.38% (0.02%)
Eagle2 `--expectIBDcM=4`	0.87% (0.03%)	0.65% (0.02%)	0.54% (0.02%)	0.39% (0.02%)

Li-Stephens	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--expectIBDcM=-0.5`	0.88% (0.03%)	0.66% (0.02%)	0.54% (0.02%)	0.39% (0.02%)
Eagle2 `--expectIBDcM=-1`	0.87% (0.02%)	0.65% (0.02%)	0.53% (0.02%)	0.39% (0.02%)
Eagle2 `--expectIBDcM=-2`	0.87% (0.03%)	0.65% (0.02%)	0.54% (0.02%)	0.39% (0.02%)
Eagle2 `--expectIBDcM=-4`	0.89% (0.03%)	0.67% (0.02%)	0.55% (0.02%)	0.40% (0.02%)

(b) CPU time per target genome

Coalescent-based	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--expectIBDcM=0.5`	1.7 min	1.7 min	1.7 min	1.8 min
Eagle2 `--expectIBDcM=1`	1.6 min	1.6 min	1.6 min	1.6 min
Eagle2 `--expectIBDcM=2` (default)	1.6 min	1.6 min	1.6 min	1.6 min
Eagle2 `--expectIBDcM=4`	1.5 min	1.4 min	1.5 min	1.5 min

Li-Stephens	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--expectIBDcM=-0.5`	2.1 min	2.1 min	2.0 min	2.1 min
Eagle2 `--expectIBDcM=-1`	1.9 min	1.9 min	1.9 min	1.9 min
Eagle2 `--expectIBDcM=-2`	1.8 min	1.8 min	1.8 min	1.8 min
Eagle2 `--expectIBDcM=-4`	1.7 min	1.7 min	1.7 min	1.7 min

We assessed the sensitivity of Eagle2 to its assumed recombination probability model (Supplementary Note Sec. 4) in our reference-based phasing benchmarks using reference panels generated from N_{ref} = 15K–100K UK Biobank samples; see Supplementary Table 3 for details. The `--expectIBDcM` parameter determines the recombination probability model: positive values invoke the SMC model [34] in which the IBD segment length distribution falls off cubically with segment length [35], while negative values invoke the Li-Stephens [21] exponential decay.

These results indicate that Eagle2's performance is insensitive to both the form (coalescent-based vs. Li-Stephens) and parameterization (expected IBD length) of the recombination probability model across a large range of reference sample sizes. We note that the slight increase in running time when using Li-Stephens probabilities is due to the cost of computing exponentials; this cost could be reduced by using lookup tables.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 12. Sensitivity of Eagle2 to its genotype error parameter.

(a) Mean switch error rate (s.e.m.)

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--genoErrProb=0.0003`	0.86% (0.02%)	0.64% (0.02%)	0.53% (0.02%)	0.37% (0.02%)
Eagle2 `--genoErrProb=0.001`	0.86% (0.02%)	0.64% (0.02%)	0.53% (0.02%)	0.38% (0.02%)
Eagle2 `--genoErrProb=0.003` (default)	0.87% (0.03%)	0.65% (0.02%)	0.53% (0.02%)	0.38% (0.02%)
Eagle2 `--genoErrProb=0.01`	0.88% (0.03%)	0.67% (0.02%)	0.55% (0.02%)	0.41% (0.02%)
Eagle2 `--genoErrProb=0.03`	0.93% (0.03%)	0.71% (0.03%)	0.59% (0.03%)	0.44% (0.02%)

(b) CPU time per target genome

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--genoErrProb=0.0003`	1.8 min	1.7 min	1.8 min	1.8 min
Eagle2 `--genoErrProb=0.001`	1.6 min	1.7 min	1.7 min	1.9 min
Eagle2 `--genoErrProb=0.003` (default)	1.6 min	1.6 min	1.6 min	1.6 min
Eagle2 `--genoErrProb=0.01`	1.4 min	1.4 min	1.4 min	1.4 min
Eagle2 `--genoErrProb=0.03`	1.4 min	1.2 min	1.3 min	1.4 min

We assessed the sensitivity of Eagle2 to its genotype error parameter ϵ (Supplementary Note, equation (6)) in our reference-based phasing benchmarks using reference panels generated from N_{ref} = 15K–100K UK Biobank samples; see Supplementary Table 3 for details. By default, Eagle2 sets $\epsilon = 0.003$.

These results indicate that Eagle2's performance is insensitive to ϵ within the range ϵ = 0.0003–0.01. We note that the reason the running time increases with decreasing ϵ is that this parameter also governs a threshold used to prune low-probability diplotypes (Supplementary Note Sec. 2.3.2); smaller values of ϵ result in a more exhaustive search through diplotype space.

Nature Genetics: doi:10.1038/ng.3679

Supplementary Table 13. Sensitivity of Eagle2 to its history length parameter.

(a) Mean switch error rate (s.e.m.)

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--histFactor=0.5`	0.92% (0.02%)	0.70% (0.02%)	0.59% (0.02%)	0.43% (0.02%)
Eagle2 `--histFactor=1` (default)	0.87% (0.03%)	0.65% (0.02%)	0.53% (0.02%)	0.38% (0.02%)
Eagle2 `--histFactor=2`	0.87% (0.03%)	0.64% (0.02%)	0.53% (0.02%)	0.38% (0.02%)
Eagle2 `--histFactor=4`	0.88% (0.03%)	0.65% (0.02%)	0.53% (0.02%)	0.38% (0.02%)

(b) CPU time per target genome

Method	N_{ref} = 15K	N_{ref} = 30K	N_{ref} = 50K	N_{ref} = 100K
Eagle2 `--histFactor=0.5`	1.5 min	1.5 min	1.5 min	1.5 min
Eagle2 `--histFactor=1` (default)	1.6 min	1.6 min	1.6 min	1.6 min
Eagle2 `--histFactor=2`	1.7 min	1.6 min	1.7 min	1.7 min
Eagle2 `--histFactor=4`	1.7 min	1.7 min	1.6 min	1.8 min

We assessed the sensitivity of Eagle2 to its history length parameter (Supplementary Note Sec. 2.2.1) in our reference-based phasing benchmarks using reference panels generated from N_{ref} = 15K–100K UK Biobank samples; see Supplementary Table 3 for details. By default, Eagle2 compares target genotypes to reference haplotypes across spans of up to 100 target hets (`--histFactor=1`). The parameter settings `--histFactor=0.5, 2, 4` change this maximum span to 50, 200, or 400 target hets, respectively.

These results indicate that Eagle2's performance is insensitive to increasing the history length parameter beyond 1. We note that the reason the running time increases only slightly with history length is that haplotype matches spanning hundreds of hets are rare. When no long matches exist, increasing the history length does not result in any extra work.

Nature Genetics: doi:10.1038/ng.3679