# Practical probabilistic and graphical formulations of long-read polyploid haplotype phasing

Jim Shaw and Yun William Yu

University of Toronto
jshaw@math.toronto.edu, ywyu@math.toronto.edu

**Abstract.** Resolving haplotypes in polyploid genomes using phase information from sequencing reads is an important and challenging problem. We introduce two new mathematical formulations of polyploid haplotype phasing: (1) the min-sum max tree partition (MSMTP) problem, which is a more flexible graphical metric compared to the standard minimum error correction (MEC) model in the polyploid setting, and (2) the uniform probabilistic error minimization (UPEM) model, which is a probabilistic analogue of the MEC model. We incorporate both formulations into a long-read based polyploid haplotype phasing method called *flopp*. We show that flopp compares favorably to state-of-the-art algorithms—up to 30 times faster with 2 times fewer switch errors on 6x ploidy simulated data.

**Keywords:** haplotype phasing · polyploid · long-reads · UPEM · MSMTP

## 1 Introduction

As genomic sequencing technologies continue to improve, we are increasingly able to resolve ever finer genomic details. Case in point, traditional genotyping only determines if a particular allele is present in a genome [32]. However, when organisms are *polyploid* (and most eukaryotic organisms are), they have multiple copies of each chromosome. We are then additionally interested in the problem of resolving *haplotypes*, i.e. determining the sequence of alleles on each specific chromosome and not just the presence of an allele within the genome. *Phasing* is the procedure of resolving the haplotypes by linking alleles within a chromosome [7].

We focus on phasing polyploid organisms using third-generation sequencing data. Many plants have *ploidy* greater than two (i.e. have more than two copies of each chromosome), such as tetraploid potatoes (Solanum tuberosum) or hexaploid wheat and cotton. Haplotype phasing has been used to gain insights into evolution [13], breeding [30], and genome-wide association studies [22], among other applications.

The most common way of determining haplotypes is to use pooled genotyping information from a population to estimate haplotypes [7]. For unrelated individuals, sophisticated statistical methods are used to determine the most likely

haplotypes for each individual [8,17,12] in the population. For related individuals, identity-by-descent information can be used for haplotype phasing [14,27]. However, these types of methods do not work on single individuals because they rely on having population data available.

Instead, in this work, we adopt the approach of single individual phasing by sequencing, which is now common in phasing human haplotypes [9]. We focus on using sequencing information for phasing, which allows us to phase a single individual without population information or prior haplotype knowledge. This is closely related to genome assembly where overlapping reads are stitched together [28]; in our case, nearby heterozygous alleles are stitched together by read information. For the rest of the paper, we use the term phasing to mean single individual phasing using sequencing information.

### 1.1 Related work

The first method for phasing polyploid genomes was HapCompass [2], which uses a graphical approach. Popular methods that followed include HapTree [4,5], H-PoP [36] and SDhaP [11]. HapTree and H-PoP heuristically maximize a likelihood function and an objective function based on the MEC model respectively while SDhaP takes a semi-definite programming approach. HapTree-X [5] additionally incorporates long-range expression correlations to allow phasing even of pairs of variants that cannot be covered by a single read, overcoming some of the problems with short-read phasing.

Due to the increased prevalence of long-read data from Oxford Nanopore or PacBio, newer methods taking advantage of the longer-range correlations accessible through long-read data have been proposed [1,33]. Unfortunately, because the error profiles of long-read technologies differ considerably from Illumina short-reads (e.g. a higher prevalence of indel errors compared to SNPs), methods tailored to short-reads such as [25,34] may be ineffective, so altogether new paradigms are required.

At a more theoretical level, in the diploid setting, the standard minimum error correction (MEC) model [6] has proven to be quite powerful. It is known to be APX-Hard and NP-Hard but heuristically solved in practice with relatively good results. Unfortunately, a good MEC score may not imply a good phasing when errors are present [21]. This shortcoming is further exacerbated in the polyploid setting because similar haplotypes may be clustered together since the MEC model does not consider coverage; this phenomenon is known as genome collapsing [33]. Thus, although the MEC model can be applied to the polyploid setting, it may be suboptimal; however, there is yet to be an alternative commonly agreed upon formulation of the polyploid phasing problem. Indeed, this is reflected in the literature: the mathematical underpinnings of the various polyploid phasing algorithms are very diverse.

### 1.2 Contributions

In this paper, we first address the theoretical shortcomings highlighted above by giving two new mathematical formulations of polyploid phasing. We adopt a

probabilistic framework that allows us to (1) give a better notion of haplotype similarity between reads and (2) define a new objective function, the uniform probabilistic error minimization (UPEM) score. Furthermore, we introduce the idea of framing the polyploid phasing problem as one of partitioning a graph to minimize the sum of the max spanning trees within each cluster, which we show is related to the MEC formulation in a specific case.

We argue that these formulations are better suited for polyploid haplotype phasing using long-reads. In addition to our theoretical justifications, we also implemented a method we call flopp (fast local polyploid phaser). flopp optimizes the UPEM score and builds up local haplotypes through the graph partitioning procedure described. When tested on simulated data sets, flopp produces much more accurate local haplotype blocks than other methods and also frequently produces the most accurate global phasing. flopp's runtime is additionally comparable to, and often much faster than, its competitors.

The code for flopp is available at https://github.com/bluenote-1577/flopp. flopp utilizes Rust-Bio [19], is written entirely in the rust programming language and is fully parallelizable. flopp takes as input either BAM + VCF files, or the same fragment file format used by AltHap [16] and H-PoP.

## 2 Methods

### 2.1 Definitions

We represent every read as a sequence of variants (rather than as a string of nucleotides, which is commonly used for mapping/assembly tasks). Let $R$ be the set of all reads that align to a chromosome and $m$ be the number of variants in our chromosome. Assuming that tetra-allelic SNPs are allowed, every read $r_i$ is considered as an element of the space $r_i \in \{-, 0, 1, 2, 3\}^m$. A read in this variant space is sometimes called a fragment. Denoting $r_i[j]$ as the $j$th coordinate of $r_i$, $r_i[j] \in \{0, 1, 2, 3\}$ if the $j$th variant is contained in the read $r_i$ where 0 represents the reference allele, 1 represents the first alternative allele, and so forth. $r_i[j] = -$ if $r_i$ does not contain the $j$th allele.

We note that flopp by default only uses SNP information, but the user may generate their own fragments, permitting indels and other types of variants to be used even if there are more than four possible alleles. The formalism is the same regardless of the types of variants used or the number of alternative alleles.

For any two reads $r_1, r_2$, let

$$d(r_1, r_2) = |\{k : r_1[k] \neq r_2[k], (r_1[k] \neq -) \wedge (r_2[k] \neq -)\}|$$

$$s(r_1, r_2) = |\{k : r_1[k] = r_2[k], (r_1[k] \neq -) \wedge (r_2[k] \neq -)\}|.$$

$d$ and $s$ stand for *different* and *same*, representing the number of different and same variants respectively between two reads.

We use $k$ to denote the ploidy. Given a $k$-ploid organism, a natural approach to phasing is to partition $R$ into $k$ subsets where the cluster membership of a read represents which haplotype the read belongs to. Let $R_1, \ldots, R_k$ be a partition of $R$. Given a partition $P = \{R_1, \ldots, R_k\}$, we denote $P[i] = R_i$.

Define the *consensus haplotype* $H(R_i) \in \{-,0,1,2,3\}^m$ associated to a subset of reads as follows. For all indices $l = 1,\ldots,m$ let $H(R_i)[l] = \arg\max_a |\{r \in R_i : r[l] = a\}|$ and break ties according to some arbitrary order. If only $-$ appear at position $l$ over all reads, we take $H(R_i)[l] = -$. It is easy to check that $H(R_i)$ is a sequence in $\{-,0,1,2,3\}^m$ such that $H(R_i)[k] \neq -$ at indices for which some read overlaps, and $\sum_{r\in R_i} d(H(R_i), r)$ is minimized.

In our formalism, we can phrase the MEC model of haplotype phasing as the task of finding a partition $\{R_1,\ldots,R_k\}$ of $R$ such that $\sum_{i=1}^k \sum_{r_j\in R_i} d(r_j, H(R_i))$, which is called the MEC score, is minimized. For notational purposes, for a subset $R_i \subset R$, define $S(R_i) = \sum_{r\in R_i} s(H(R_i), r)$ and $D(R_i) = \sum_{r\in R_i} d(H(R_i), r)$. $\sum_{i=1}^k D(R_i)$ is just the MEC score for a particular partition.

### 2.2 Problem formulation

**Min-sum max tree partition (MSMTP) model.** Let $G(R) = (R, E, w)$ be an undirected graph where the vertices are $R$ and edges $E$ are present between two reads $r_1, r_2$ if $r_1, r_2$ overlap, i.e. $d(r_1, r_2) + s(r_1, r_2) > 0$. Let the weight of $e = (r_1, r_2)$ be $w(e) = w(r_1, r_2)$ for some weight function $w$. We call $G(R)$ the *read-graph*, see Figure 1. A similar notion is found in [11,1,33,24,23].

For a partition of $R$ into disjoint subsets $\{R_1,\ldots,R_k\}$ we take $G(R_i)$ as defined above. We only consider partitions of vertices such that all $G(R_i)$ are connected, which we will denote as valid partitions. Let $MST(G)$ be the maximum spanning tree of a graph $G$. Define

$$SMTP_R^k(R_1,\ldots,R_k) = \sum_{i=1}^{k} \sum_{e\in MST(G(R_i))} w(e). \tag{1}$$

We formulate the *min-sum max tree partition (MSMTP) problem* as finding a valid partition $\{R_1,\ldots,R_k\}$ of $R$ such that $SMTP_R^k(R_1,\ldots,R_k)$ is minimized. We refer to the *objective function being minimized* as the SMTP score and *the computational problem of minimizing the SMTP score* as the MSMTP problem.

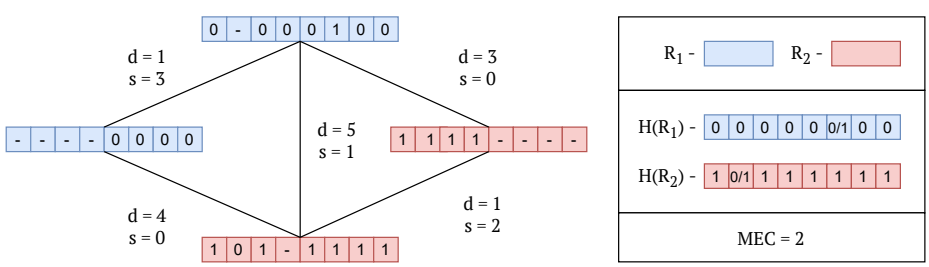


**Fig. 1.** An example of a read-graph (without the weighting $w$ specified) along with corresponding $d, s$ values between reads. The colors represent a partition of the read-graph into two subsets $R_1, R_2$. The consensus haplotypes $H(R_1), H(R_2)$ are shown where the 0/1 indicates that either 1 or 0 are valid alleles for a consensus haplotype.

The MSMTP problem falls under a class of problems called graph tree partition problems [10], most of which are NP-Hard. We give a proof that MSMTP is NP-Hard in Appendix A.

Intuitively, assuming each $G(R_i)$ is connected, a maximum spanning tree is a maximum measure of discordance along the entire haplotype. We prove below that under a specific constraint on the read-graph, the SMTP score for $w(r_1, r_2) = d(r_1, r_2)$ is an upper bound for the MEC score.

**Theorem 1.** *Suppose $w(r_1, r_2) = d(r_1, r_2)$. Let $a, b \in \mathbb{N}$ and let $R$ be a set of fragments such that for every $r \in R$, for all $k \in \{a, a+1, ..., b\}$, $r[k] \neq -$ and for $l \notin \{a, a+1, \cdots, b\}$, $r[l] = -$. For any $R_i$ in a valid partition $\{R_1, ..., R_k\}$ of $R$,*

$$\sum_{e \in MST(G(R_i))} w(e) + \min_{r \in R_i} d(r, H(R_i)) \geq \sum_{r \in R_i} d(H(R_i), r).$$

*Therefore,*

$$SMTP_R^k(R_1, ..., R_k) + \sum_{i=1}^{k} \min_{r \in R_i} d(r, H(R_i)) \geq \sum_{i=1}^{k} \sum_{r \in R_i} d(H(R_i), r).$$

*Proof.* Take the augmented graph $G(R_i \cup \{H(R_i)\})$. It is clear that $\sum_{r \in R_i} d(H(R_i), r)$ is just $\sum_{e \in Star(H(R_i))} w(e)$, where $Star(H(R_i))$ is the star-graph having internal node $H(R_i)$ and every $r \in R_i$ as a leaf node.

Now note that $H(R_i)$ is constructed *precisely* as a sequence in $\{-, 0, 1, 2, 3\}^m$ which is non $-$ at indices $a, a+1, \ldots, b$ that minimizes the sum $\sum_{e \in Star(H(R_i))} w(e)$. For any $r \in R_i$, $r$ is also non $-$ at the same indices by assumption, so $\sum_{e \in Star(r)} w(e) \geq \sum_{e \in Star(H(R_i))} w(e)$.

Removing the node $H(R_i)$ from $Star(r)$ and the corresponding edge, we get a spanning tree of $G(R_i)$; call this new graph $Star(r)'$. Thus $\sum_{e \in Star(r)'} w(e) + d(r, H(R_i)) \geq \sum_{r \in R_i} d(H(R_i), r)$, and clearly $\sum_{e \in MST(G)} w(e) \geq \sum_{e \in Star(r)'} w(e)$. The inequality holds for any $r$, so we can choose $r$ to minimize $w(r, H(R_i))$, finishing the proof.

The above theorem relies on the assumption that all reads in the set overlap exactly. While this is obviously not true for the entire genome, flopp takes a local clustering approach where the entire read set $R$ is partitioned into smaller local read sets that overlap significantly.

We verified experimentally that the SMTP score for partitions generated by our method has a strongly linear dependence on the MEC score when $w = d$, see Appendix E. These results justify that minimizing the SMTP score is worthwhile for this specific case. However, we do not have to necessarily use $w = d$. In Section 2.3 and we opt for a more theoretically sound probabilistic weighting.

**Uniform probabilistic error minimization (UPEM) model** The SMTP score has problems with collapsing genomes in the same manner the MEC score does; it does not take into account the assumption that coverage should be uniform between haplotypes. Concretely, if a polyploid organism has two identical

haplotypes, the reads from both haplotypes may be clustered together in the MEC model and a noisy read may instead be placed in its own cluster.

Let $\epsilon$ represent the probability that a variant is called incorrectly. Let $\sigma \in \mathbb{R}$ be a normalizing constant, and $X_i \sim Binomial(\lceil (D(R_i) + S(R_i))/\sigma \rceil, \epsilon)$ be a binomial random variable. Then

$$UPEM_R(R_1, \ldots, R_k) = \sum_{i=1}^{k} \log \left[ \Pr \left( X_i > \left\lceil \frac{D(R_i)}{\sigma} \right\rceil \right) \right] + \log[\chi^2(|R_1|, \ldots, |R_k|)].$$

The $\chi^2(x_1, \ldots, x_n)$ term is the p-value for the $\chi^2$ test while the binomial term is a sum of log one-sided binomial tests where the null hypothesis is that the error rate of a clustering is $\epsilon$. Therefore the UPEM score is just a sum of log p-values.

The UPEM score is a probabilistic version of the MEC model under the hypothesis that the errors and coverage are uniform across haplotypes. The parameter $\sigma$ is a normalizing constant and is important because if a specific genome has a high rate of heterozygosity and $\epsilon$ is slightly underestimated, then the sample size $D(R_i) + S(R_i)$ is large. The p-value associated with the binomial random variable will be extremely small and drown out the contributions from the $\chi^2$ term, so $\sigma$ is a learned data set specific constant used to keep the two terms balanced.

UPEM maximization enforces a relatively uniform partition. Furthermore, errors will be distributed among partitions equally due to the non-linearity of the UPEM score; if one cluster is extremely erroneous, the sum of the binomial terms may be higher for a more spread out error even if the overall MEC score is slightly higher. If error and coverage uniformity assumptions are satisfied, clearly these two properties are desirable in an objective function.

### 2.3 Local graph clustering

We now discuss the algorithms implemented in flopp. A high-level block diagram showing outlining flopp's main processing stages are outlined in Figure 2. Importantly, flopp is a local clustering method, which means that we first attempt to find good partitions for smaller subsets of reads by optimizing the SMTP and UPEM functions and then joining haplotype blocks together afterwards.

**Choice of edge weight function** Previous methods that use a read-graph formalism such as SDhaP [11], FastHap [24], WHP [33], and nPhase [1] all define different weightings between reads. Two previously used weight functions are

$$w_{SDhaP}(r_1, r_2) = \frac{s(r_1, r_2) - d(r_1, r_2)}{s(r_1, r_2) + d(r_1, r_2)} \text{ (SDhaP)}$$

$$w_{error}(r_1, r_2) = \frac{s(r_1, r_2)}{s(r_1, r_2) + d(r_1, r_2)} \text{ (nPhase; FastHap is similar).}$$

These weightings are quite simple and have issues when the length of the reads have high variance, as is the case with long-reads. A more sophisticated approach is to use a probabilistic model. Let $E(r_1, r_2) = 1 - w_{error}(r_1, r_2)$ be

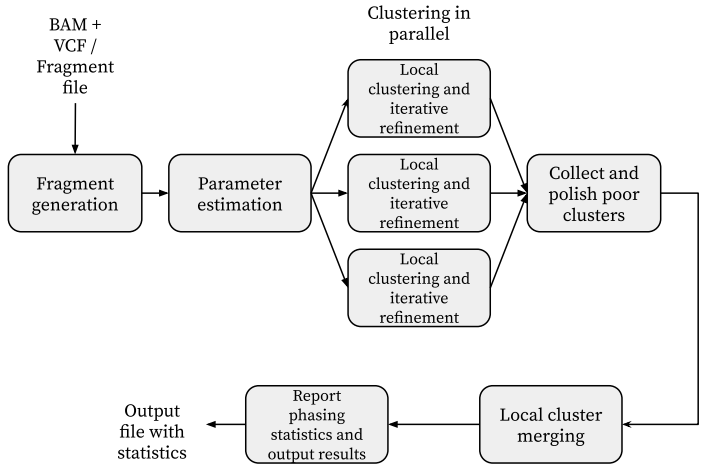


**Fig. 2.** A block diagram outlining flopp's major processing steps. flopp takes BAM+VCF files or a fragment file as input. The fragments are clustered into local haplotype blocks, which are then polished and merged. The final phasing is output to the user as well as various phasing statistics.

the average error rate between two reads. Assuming an error parameter $\epsilon$ as described in the UPEM formula, we define

$$D_{KL}(r_1, r_2||\epsilon) = E(r_1, r_2) \log\left(\frac{E(r_1, r_2)}{2\epsilon(1-\epsilon)}\right) + (1 - E(r_1, r_2)) \log\left(\frac{1 - E(r_1, r_2)}{1 - 2\epsilon(1-\epsilon)}\right).$$

$D_{KL}(r_1, r_2||\epsilon)$ is the Kullback-Leibler divergence between a $E(r_1, r_2)-$coin and a $2\epsilon(1-\epsilon)-$coin. The reason we use $2\epsilon(1-\epsilon)$ is because, given two reads with error rate $\epsilon$, the probability that two variants from the same haplotype are different is $2\epsilon(1-\epsilon)$. Now let

$$w(r_1, r_2) = [s(r_1, r_2) + d(r_1, r_2)] \cdot D_{KL}(r_1, r_2||\epsilon). \tag{2}$$

$D_{KL}(r_1, r_2||\epsilon)$ is a measure of divergence between the error profiles of the two reads and the expected error profile for two reads from the same haplotype. There is a slight issue that $D_{KL}(r_1, r_2||E)$ has a minimum at $E(r_1, r_2) = 2\epsilon(1-\epsilon)$ when $\epsilon$ is fixed. We want $D_{KL}$ to be a monotonically increasing function with respect to $E$, so we flip the sign of $D_{KL}(r_1, r_2||\epsilon)$ if $E < 2\epsilon(1-\epsilon)$.

There is a nice interpretation of $w(r_1, r_2)$ when $E(r_1, r_2) > 2\epsilon(1-\epsilon)$ as given by the following theorem.

**Theorem 2 (Arratia and Gordon [3]).**
*Suppose $p < a < 1$. Let $H = a \log \frac{a}{p} + (1-a) \log(\frac{1-a}{1-p})$ be the Kullback-Leibler divergence between an $a-$coin and a $p-$coin. Then*

$$\Pr(Binom(n, p) \geq an) \leq e^{-nH}. \tag{3}$$

*In particular, this bound is asymptotically tight up to logarithms, i.e.*

$$\lim_{n\to\infty} \frac{\log \Pr(Binom(n,p) \geq an)}{-nH} = 1. \tag{4}$$

The proof of the above theorem is simple; it follows by an application of a Chernoff bound and Sterling's inequality. In [3], they show that this approximation is particularly good when $a >> p$. Theorem 2 gives an interpretation of $w(r_1, r_2)$ as the negative log value of a 1-sided binomial test where the null hypothesis is that $r_1, r_2$ come from the same haplotype, the number of trials is $s(r_1, r_2) + d(r_1, r_2)$ and the number of successes is $d(r_1, r_2)$.

WHP [33] uses a log ratio of binomial probability densities as their edge weight. Mathematically, the resulting formula is almost the same as Equation 2, except they fix $E(r_1, r_2)$ to be the average error rate between reads from different haplotypes, a constant. They end up with both negative and positive edges with large magnitude and Theorem 2 does *not* apply to their case. On the other hand, our edge weights are rigorously justified as negative log p-values for a binomial test, making it more interpretable.

**MSMTP Algorithm** We devised a heuristic algorithm for local clustering inspired by the MSMTP problem. From now on, let $S \subset R$. The pseudocode is shown in Algorithm 1. In the diploid setting, the main idea behind this algorithm is similar to that of FastHap [24], but the algorithm is quite different after generalizing to higher ploidies.

**Algorithm 1:** Greedy min-max read partitioning

**Input :** Read-graph $G(S)$, ploidy $k$, iterations $n$
**Output:** A partition $\{S_1, \ldots, S_k\}$ of $S$

```
{v_1, ..., v_k} ← FindMaxClique(G(S), k)
for i = 1 to k do
    S_i ← {v_i}
end
for i = 1 to n do
    V ← G(S) \ ⋃_{i=1}^{k} S_i
    Reverse-sort V by assigning to v ∈ V the value
      v → min_{S_i ∈ {S_1,...,S_k}} max_{r ∈ S_i} s(r, v) + d(r, v)
    V ← V[: ⌈|V|/n⌉]
    for v in V do
        S' ← arg min_{S_i} max_{r ∈ S_i} w(v, r)
        S' ← S' ∪ {v}
    end
end
Return {S_1, ..., S_k}
```

For the FindMaxClique method mentioned in Algorithm 1, we use a greedy clique-finding method which takes $O(k|S| \log |S| + k^2|S|)$ time. First, we take the

heaviest edge between two vertices and make this our 2-clique. Then we re-sort vertices by maximizing the minimum distance from the vertex to all vertices in the 2-clique, add a third edge to get a 3-clique, and so forth until we get a $k-$clique.

The complexity of the local clustering algorithm is $O(n|S|^2 + k|S| \log |S| + k^2|S|)$. In practice, note that $|S| >> k$. The parameter $n$ is fixed to be 10. By iterating over $n$, we re-sort the edges based on their overlaps to the new clusters, which have changed since the previous iteration. This improves the order in which we add vertices to the clusters.

The connection to MSMTP is at line 10. A priori, it is not obvious what metric to use to determine which cluster to put the vertex in. For Kruskal's algorithm, one starts by considering the heaviest edges, so we decided to minimize the maximum edge from the vertex to the cluster so that the heaviest edges are small. Intuitively, a maximum spanning tree is sensitive to a highly erroneous edge, so we prioritize minimizing the worst edge even if on average the vertex connects to the cluster well.

**Iterative refinement of local clusters** We refine the output of the local clustering procedure by optimizing the UPEM score using a method similar to the Kernighan-Lin algorithm [18]. Pseudocode can be found in the appendix as Algorithm 2.

The algorithm takes in a partition $P = \{S_1, \ldots, S_k\}$ of a subset of reads $S \subset R$ as well as a parameter $n$ which indicates the maximum number of iterations. The algorithm checks how moving reads from one partition to another partition changes the UPEM score for every read. We store the best moves and execute a fraction of them. Proceed for $n$ iterations or until the UPEM score does not improve anymore. In practice, we set the parameter $n = 10$. The time complexity of Algorithm 2 is $O(n|S|k \log(|S|k))$.

**Local phasing procedure** Note that Algorithms 1 and 2 work on subsets of reads or subgraphs of the underlying read-graph. Let $b \in \mathbb{N}$ be a constant representing the length of a local block. We consider subsets $B_1, \ldots, B_l \subset R$ where

$$B_i = \{r \in R : \exists j, b \cdot (i-1) \leq j \leq b \cdot i, r[j] \neq -\}.$$

The subsets are just all reads that overlap a shifted interval of size $b$, similar to the work done in [31]. After choosing a suitable $b$, we run the read-partitioning and iterative refinement on all $B_1, \ldots, B_l$ to generate a set of partitions $P_1, \ldots, P_l$. We found that a suitable value of $b$ is the $\frac{1}{3}-$quantile value of read lengths. By read length we mean the last non $'-'$ position minus the first non $'-'$ position of $r \in \{0, 1, 2, 3, -\}$.

It is important to note that computationally, the local clustering procedure is easily parallelizable. The local clustering step has therefore a linear speed-up in the number of threads.

### 2.4 Polishing, merging, and parameter estimation

**Filling in erroneous blocks** Once we obtain a set of partitions $P_1, \ldots, P_l$ of $B_1, \ldots, B_l$ according to the above local clustering procedure, we can identify partitions with low UPEM score and correct them. We describe this procedure in Appendix C.

**Local cluster merging** Let $P$ represent the final partition of all reads $R$. We build $P$ given $P_1, \ldots, P_l$ as follows. Start with $P = P_1$. Let $S_k$ be the symmetric group on $k$ elements, i.e. the set of all permutations on $k-$elements. At the $i$th step of the procedure, let

$$\sigma_i = \arg\max_{\sigma \in S_k} \sum_{j=1}^{k} |P[j] \cap P_{i+1}[\sigma(j)]|.$$

Then let $P[j] = P[j] \cup P_{i+1}[\sigma_i(j)]$ for all $j = 1, \ldots, k$. Repeat this procedure for $i = 1, \ldots, l-1$.

We experimented with more sophisticated merging procedures such as a beam-search approach but did not observe a significantly better phasing on any of our data sets.

**Phasing output** Once a final partition $P$ has been found, we build the final phased haplotypes. The output is $k$ sequences in $\{0, 1, 2, 3, -\}^m$ where $m$ is the number of variants. flopp can take fragment files as input, in which each line of the file describes a single read fragment in $\{0, 1, 2, 3, -\}^m$, or it can take a VCF file and a BAM file of aligned reads. In the former case, without a VCF file, we do not have genotyping information, so we simply output $\{H_1, \ldots, H_k\}$ where $H_i = H(P[i])$ is the consensus haplotype. If a VCF file is present, flopp constrains the final haplotype by the genotypes, that is, for some output variant, the number of reference and alternate alleles is the same as in the VCF file. We describe this procedure in Appendix D.

**Parameter estimation** We already mentioned in previous sections how we set all parameters for algorithms except for $\epsilon$ and the parameter $\sigma$ in the UPEM score. We set $\sigma$ to be the median length of the reads divided by 25, which we found empirically to be a good balance between the binomial and chi-squared terms.

To estimate $\epsilon$, we start with an initial guess of $\epsilon = 0.03$. We then select 10 subsets $B_i \subset R$ at random and perform local clustering and refinement. We estimate $\epsilon$ from the $10 \cdot k$ total clusters by choosing $\epsilon =$ the $\frac{1}{10}$th quantile error, where for each cluster $C \subset B_i$ the error is $\frac{D(C)}{S(C)+D(C)}$. We pick a bottom quantile because we assume that there is some error in our method, so to get the true error rate we must underestimate the computed errors.

## 3 Results and Discussion

### 3.1 Phasing metrics

There are a plethora of phasing metrics developed for diploid and polyploid phasing [26]. We use three different metrics of accuracy, but argue that each

individual metric can be misleading and that all three metrics should be used in unison before drawing conclusions on phasing accuracy.

For a global measure of goodness of phasing, we use the Hamming error rate. Given a set of true haplotypes $H = \{H[1], \ldots, H[k]\}$ and a set of candidate haplotypes $H^* = \{H^*[1], \ldots, H^*[k]\}$, we define the Hamming error rate as

$$\mathrm{HE}(H, H^*) = \min_{\sigma \in S_k} \sum_{i=1}^{k} d'(H[i], H^*[\sigma(i)])/mk$$

where $S_k$ is the set of permutations on $k$ elements, $m$ is the length of each $H[i]$, $k$ is the ploidy, and $d'$ is the same as the $d$ function defined before except we count the case where one haplotype has a $'-'$ at a coordinate as an error.

We define the switch error rate (SWER) similarly to WHP [33]. Let $\Pi_i \subset S_k$ be the set of permutations such that $H[j][i] = H[j][\sigma(i)]$ for all $1 \leq j \leq k$. These are the mappings from the truth to the candidate haplotypes that preserve the alleles at position $i$. Then we define the switch error as

$$\mathrm{SWER}(H, H^*) = \min_{\sigma_1 \in \Pi_1, \ldots, \sigma_{n-1} \in \Pi_{n-1}} \frac{1}{n} \sum_{i=1}^{n-1} 1_{\sigma_i \neq \sigma_{i+1}}$$

where $1_{\sigma_i \neq \sigma_{i+1}} = 1$ if $\sigma_i \neq \sigma_{i+1}$ and 0 otherwise.

The Hamming error, while easily interpretable, can be unstable. A single switch error can drastically alter the Hamming error rate. The SWER also has issues; for example, if two switches occur consecutively, the phasing is still relatively good but the SWER is worse than if only one switch occurred.

We define a new error rate, called the q-block error rate. For a haplotype $H[i]$, break $H[i]$ into non-overlapping substrings of length $q$. Denote each new block $H[i]^1, \ldots, H[i]^{\ell_q}$. For a set of haplotypes $H$, doing this for every haplotype gives a collection of haplotype blocks $H^1, \ldots, H^{\ell_q}$. Then the q-block error rate is

$$\text{q-block}(H, H^*) = \frac{1}{\ell_q} \sum_{i=1}^{\ell_q} \mathrm{HE}(H^i, H^{i*}).$$

The q-block error rate measures local goodness of assembly and interpolates between the Hamming error rate, when $q$ is the size of the genome, and a metric similar to the switch error when $q = 2$.

### 3.2 Simulation procedure

We used the v4.04 assembly of Solanum tuberosum produced by the Potato Genome Sequencing Consortium [15] as a reference. We took the first 3.5 Mb of chromosome 1 and removed all "N"s, leaving about 3.02 Mb and simulated bi-allelic SNPs using the haplogenerator software [26], a script made specifically for realistic simulation of variants in polyploid genomes.

We generated SNPs with mean distance between SNPs of 45 bp. This is in line with the 42.5 bp average distance between variants as seen in [35] for Solanum tuberosum; in that study they observe that $> 80\%$ of variants are bi-allelic SNPs. This is also in line with the 58 bp mean distance between variants

seen for the hexaploid sweet potato (Ipomoea batatas) observed in [38]. The dosage probabilities provided to haplogenerator are the same parameters as used in [26] for the tetraploid case. When simulating triploid genomes, we use the same dosage probabilities but disallow the case for three alternate alleles. We repeated our test for mean distance between SNPs of 90 and 135 bp in Appendix G.

We used two different software packages for simulating reads. We used PaSS [39] with the provided default error profiles for simulating PacBio RS reads. PaSS has higher error rates than other methods like PBSIM [29] which tends to underestimate error [39]. We used NanoSim [37] for simulating nanopore reads using a default pre-trained error model based on real human dataset provided by the software.

After generating the haplotypes and simulating the reads from the haplotypes, we obtain a truth VCF from the haplotypes. We map the reads using minimap2 [20] to the reference genome. The scripts for the simulation pipeline can be found at https://github.com/bluenote-1577/flopp_test.

### 3.3 Results on simulated data set

We primarily test against H-PoPG [36], the genotype constrained version of H-PoP. We tried testing against WHP, but found that the results for WHP were extremely discontiguous and the accuracy was relatively poor across our data sets. We show an example of this in Appendix F. Other methods such as HapTree and AltHap were tested, but we ran into issues with either computing time or poor accuracy due to the methods not being suited for long-read data. We did not test against nPhase because the output of nPhase does not have a fixed number of haplotypes.

The switch error and Hamming error rates are shown in Figure 3. For each test, we ran the entire pipeline three times; each run at high ploidies takes on the timescale of days to complete. The run times on PacBio reads for H-PoPG, flopp, as well as one instance of WHP on 3x ploidy data are shown in Figure 4.

For the nanopore data set simulated from NanoSim, the results for H-PoPG and flopp are very similar. The PaSS PacBio simulator outputs reads that are more erroneous, and we can see that flopp generally performs better than H-PoPG across ploidies except for the Hamming error rate when the coverage is relatively low; interestingly, the switch error rate is still lower in this case. flopp's switch error rate is consistently 1.5-2 times lower than H-PoPG for the simulated PacBio reads. On the low coverage data sets, H-PoPG's global optimization strategy leads to a better global phasing than flopp's local methodology.

Note that in these tests we phase a contig of length 3.02 Mb, whereas most genes of interest are $< 50$ kb. In Figure 5 we plot the mean q-block error rates of the 5x and 6x ploidy phasings at 10x coverage; these runs have higher Hamming error rates for flopp than H-PoPG. For blocks of up to around $850 \cdot 45 = 38250$ bases, flopp outputs phasings with lower q-block error rates than H-PoPG despite a larger global error rate. Although flopp may sometimes give worse global phasings than H-PoPG, flopp can give extremely accurate local phasings.

Computationally, Figure 4 shows that flopp is at least 3 times faster than H-PoPG and up to 30 times faster than H-PoPG for 6x ploidy on a *single thread*.

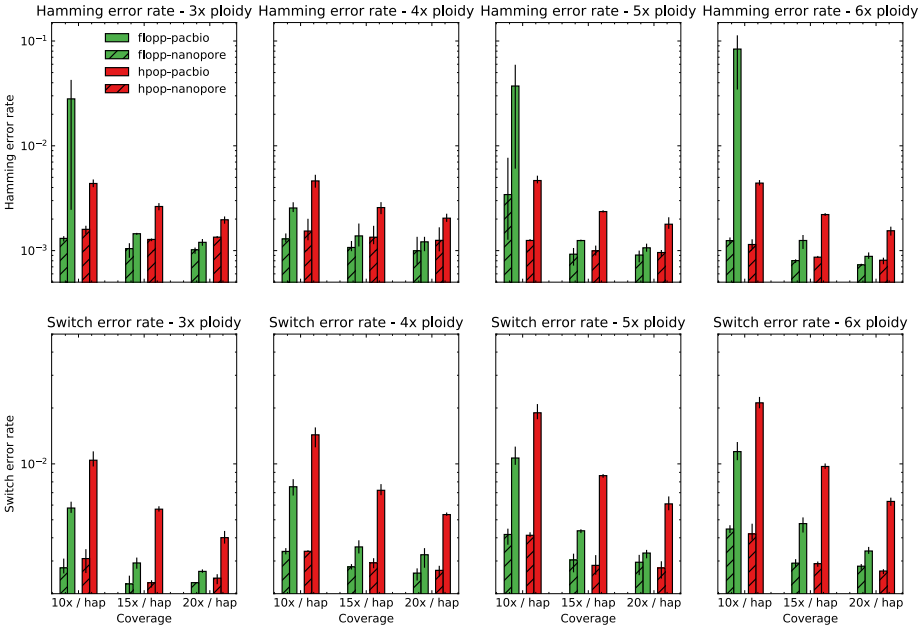


**Fig. 3.** The mean switch error rate and Hamming error rate from testing on simulated data sets as described in Section 3.2 over a range of ploidies and coverages on two different types of read simulators. The error bars represent the lowest and highest values for the metric over three iterations. The results on the Nanopore simulated reads from NanoSim [37] are similar for both methods. flopp achieves much better switch error rates on the PacBio simulated reads from PaSS [39], although for low coverage flopp sometimes has a higher Hamming error rate.

The local clustering step sees a linear speedup from multi-threading. For most data sets, we get a 3-4 times speedup with 10 threads.

## 4 Conclusion

In this paper, we presented two new formulations of polyploid phasing, the min-sum max tree partition (MSMTP) problem and the uniform probabilistic error minimization (UPEM) model. The SMTP score is a flexible graphical interpretation of haplotype phasing which is related to the MEC score when using a specific weighting on the read-graph, whereas the UPEM score is a superior version of the MEC score when uniformity assumptions are satisfied. Using our probabilistic formulation, we give a new notion of distance between read fragments based on the Kullback-Leibler divergence which has a rigorous interpretation as a log p-value of a one-sided binomial test.

We implemented a fast, local phasing procedure using these new formulations and showed that our software, flopp, is faster and more accurate on high coverage data while always having extremely accurate local phasings across a range of error profiles, coverages, and ploidies.

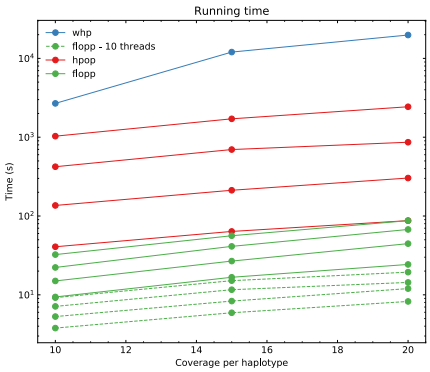


**Fig. 4.** Run times for flopp, H-PoPG, and one instance of WHP. Each line represents a different ploidy with higher ploidies taking longer. The one instance of WHP was run at 3x ploidy.

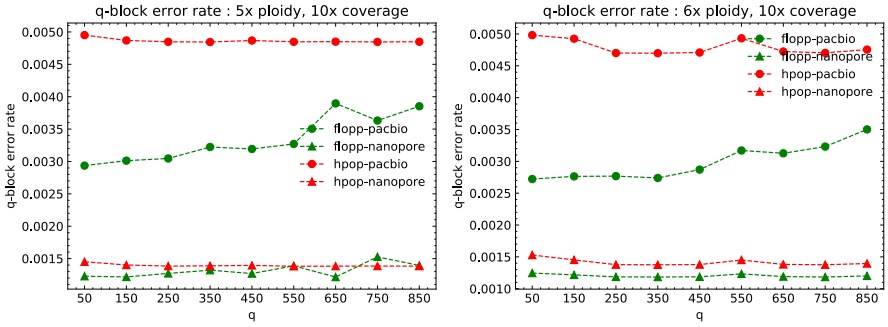


**Fig. 5.** The mean q-block error rates for the 5x and 6x ploidy data sets at 10x coverage per haplotype over three iterations. Each variant is spaced on average 45 bp apart, so each block is of size $\sim q \cdot 45$. The upward trend indicates an inaccurate global phasing, but the q-block error rates for flopp are lower than for H-PoP in this regime.

**Acknowledgments.** This work was supported by a faculty startup fund from the University of Toronto at Scarborough Department of Computer and Mathematical Sciences.

## References

1. Abou Saada, O., Tsouris, A., Friedrich, A., Schacherer, J.: nPhase: An accurate and contiguous phasing method for polyploids. bioRxiv p. 2020.07.24.219105 (2020)
2. Aguiar, D., Istrail, S.: HapCompass: A Fast Cycle Basis Algorithm for Accurate Haplotype Assembly of Sequence Data. Journal of Computational Biology **19**(6), 577–590 (2012)
3. Arratia, R., Gordon, L.: Tutorial on large deviations for the binomial distribution. Bulletin of Mathematical Biology **51**(1), 125–131 (1989)

4. Berger, E., Yorukoglu, D., Peng, J., Berger, B.: HapTree: A Novel Bayesian Framework for Single Individual Polyplotyping Using NGS Data. PLoS Computational Biology **10**(3), e1003502 (2014)
5. Berger, E., Yorukoglu, D., Zhang, L., Nyquist, S.K., Shalek, A.K., Kellis, M., Numanagić, I., Berger, B.: Improved haplotype inference by exploiting long-range linking and allelic imbalance in RNA-seq datasets. Nature Communications **11**(1), 4662 (2020)
6. Bonizzoni, P., Dondi, R., Klau, G.W., Pirola, Y., Pisanti, N., Zaccaria, S.: On the Minimum Error Correction Problem for Haplotype Assembly in Diploid and Polyploid Genomes. Journal of Computational Biology: A Journal of Computational Molecular Cell Biology **23**(9), 718–736 (2016)
7. Browning, S.R., Browning, B.L.: Haplotype phasing: existing methods and new developments. Nature Reviews Genetics **12**(10), 703–714 (2011)
8. Browning, S., Browning, B.: 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**(5), 1084–1097 (2007)
9. Choi, Y., Chan, A.P., Kirkness, E., Telenti, A., Schork, N.J.: Comparison of phasing strategies for whole human genomes. PLoS Genetics **14**(4) (2018)
10. Cordone, R., Maffioli, F.: On the complexity of graph tree partition problems. Discrete Applied Mathematics **134**(1), 51–65 (2004)
11. Das, S., Vikalo, H.: SDhaP: haplotype assembly for diploids and polyploids via semi-definite programming. BMC Genomics **16**(1), 260 (2015)
12. Delaneau, O., Zagury, J.F., Robinson, M.R., Marchini, J.L., Dermitzakis, E.T.: Accurate, scalable and integrative haplotype estimation. Nature Communications **10**(1), 5436 (2019)
13. Eriksson, J.S., de Sousa, F., Bertrand, Y.J.K., Antonelli, A., Oxelman, B., Pfeil, B.E.: Allele phasing is critical to revealing a shared allopolyploid origin of Medicago arborea and M. strasseri (Fabaceae). BMC Evolutionary Biology **18**(1), 9 (2018)
14. Gao, G., Allison, D.B., Hoeschele, I.: Haplotyping Methods for Pedigrees. Human Heredity **67**(4), 248–266 (2009)
15. Hardigan, M.A., Crisovan, E., Hamilton, J.P., Kim, J., Laimbeer, P., Leisner, C.P., Manrique-Carpintero, N.C., Newton, L., Pham, G.M., Vaillancourt, B., Yang, X., Zeng, Z., Douches, D.S., Jiang, J., Veilleux, R.E., Buell, C.R.: Genome Reduction Uncovers a Large Dispensable Genome and Adaptive Role for Copy Number Variation in Asexually Propagated Solanum tuberosum[OPEN]. The Plant Cell **28**(2), 388–405 (2016)
16. Hashemi, A., Zhu, B., Vikalo, H.: Sparse Tensor Decomposition for Haplotype Assembly of Diploids and Polyploids. BMC Genomics **19**(S4), 191 (2018)
17. Howie, B.N., Donnelly, P., Marchini, J.: A Flexible and Accurate Genotype Imputation Method for the Next Generation of Genome-Wide Association Studies. PLOS Genetics **5**(6), e1000529 (2009)
18. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal **49**(2), 291–307 (1970)
19. Köster, J.: Rust-Bio: a fast and safe bioinformatics library. Bioinformatics **32**(3), 444–446 (2016)
20. Li, H.: Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics **34**(18), 3094–3100 (2018)
21. Majidian, S., Kahaei, M.H., de Ridder, D.: Minimum error correction-based haplotype assembly: Considerations for long read data. PLOS ONE **15**(6), e0234470 (2020)

22. Maldonado, C., Mora, F., Scapim, C.A., Coan, M.: Genome-wide haplotype-based association analysis of key traits of plant lodging and architecture of maize identifies major determinants for leaf angle: hapLA4. PLoS ONE **14**(3) (2019)
23. Mazrouee, S., Wang, W.: PolyCluster: Minimum Fragment Disagreement Clustering for Polyploid Phasing. IEEE/ACM Transactions on Computational Biology and Bioinformatics **17**(1), 264–277 (2020)
24. Mazrouee, S., Wang, W.: FastHap: fast and accurate single individual haplotype reconstruction using fuzzy conflict graphs. Bioinformatics **30**(17), i371–i378 (2014)
25. Moeinzadeh, M.H., Yang, J., Muzychenko, E., Gallone, G., Heller, D., Reinert, K., Haas, S., Vingron, M.: Ranbow: A fast and accurate method for polyploid haplotype reconstruction. PLOS Computational Biology **16**(5), e1007843 (2020)
26. Motazedi, E., Finkers, R., Maliepaard, C., de Ridder, D.: Exploiting next-generation sequencing to solve the haplotyping puzzle in polyploids: a simulation study. Briefings in Bioinformatics p. bbw126 (2017)
27. Motazedi, E., de Ridder, D., Finkers, R., Baldwin, S., Thomson, S., Monaghan, K., Maliepaard, C.: TriPoly: haplotype estimation for polyploids using sequencing data of related individuals. Bioinformatics **34**(22), 3864–3872 (2018)
28. Nagarajan, N., Pop, M.: Sequence assembly demystified. Nature Reviews Genetics **14**(3), 157–167 (2013)
29. Ono, Y., Asai, K., Hamada, M.: PBSIM: PacBio reads simulator—toward accurate genome assembly. Bioinformatics **29**(1), 119–121 (2013)
30. Qian, L., Hickey, L.T., Stahl, A., Werner, C.R., Hayes, B., Snowdon, R.J., Voss-Fels, K.P.: Exploring and Harnessing Haplotype Diversity to Improve Yield Stability in Crops. Frontiers in Plant Science **8** (2017)
31. Sankararaman, A., Vikalo, H., Baccelli, F.: ComHapDet: a spatial community detection algorithm for haplotype assembly. BMC Genomics **21**(9), 586 (2020)
32. Scheben, A., Batley, J., Edwards, D.: Genotyping-by-sequencing approaches to characterize crop genomes: choosing the right tool for the right application. Plant Biotechnology Journal **15**(2), 149–161 (2017)
33. Schrinner, S.D., Mari, R.S., Ebler, J., Rautiainen, M., Seillier, L., Reimer, J.J., Usadel, B., Marschall, T., Klau, G.W.: Haplotype threading: accurate polyploid phasing from long reads. Genome Biology **21**(1), 252 (2020)
34. Siragusa, E., Haiminen, N., Finkers, R., Visser, R., Parida, L.: Haplotype assembly of autotetraploid potato using integer linear programing. Bioinformatics **35**(18), 3279–3286 (2019)
35. Uitdewilligen, J.G.A.M.L., Wolters, A.M.A., D'hoop, B.B., Borm, T.J.A., Visser, R.G.F., van Eck, H.J.: A Next-Generation Sequencing Method for Genotyping-by-Sequencing of Highly Heterozygous Autotetraploid Potato. PLoS ONE **8**(5) (2013)
36. Xie, M., Wu, Q., Wang, J., Jiang, T.: H-PoP and H-PoPG: heuristic partitioning algorithms for single individual haplotyping of polyploids. Bioinformatics **32**(24), 3735–3744 (2016)
37. Yang, C., Chu, J., Warren, R.L., Birol, I.: NanoSim: nanopore sequence read simulator based on statistical characterization. GigaScience **6**(4), 1–6 (2017)
38. Yang, J., Moeinzadeh, M.H., Kuhl, H., Helmuth, J., Xiao, P., Haas, S., Liu, G., Zheng, J., Sun, Z., Fan, W., Deng, G., Wang, H., Hu, F., Zhao, S., Fernie, A.R., Boerno, S., Timmermann, B., Zhang, P., Vingron, M.: Haplotype-resolved sweet potato genome traces back its hexaploidization history. Nature Plants **3**(9), 696–703 (2017)
39. Zhang, W., Jia, B., Wei, C.: PaSS: a sequencing simulator for PacBio sequencing. BMC Bioinformatics **20**(1), 352 (2019)

## A NP-Hardness of MSMTP

Let $G = (V, E, w)$ be a connected, undirected, and weighted graph with weight function $w$. Call a partition $V_1, \ldots, V_k$ of the vertices into $k$ disjoint sets valid if each $G(V_i)$ is connected. Then the min-sum max tree partition (MSMTP) problem is to find a valid partition of $V$ into $k$ sets that minimizes $SMTP_G^k(V_1, \ldots, V_k)$.

**Theorem 3.** *MSMTP is NP-Hard for $k \geq 3$.*

*Proof.* We take a reduction from graph coloring.

Let $G' = (V, E', w')$ be a complete graph where the vertices of $G'$ are the same as $G$. Let the weight of any edge $e \in E'$ to be 2 if $e \in E$, the original graph, and let it be 1 otherwise.

Let $V_1, \ldots, V_k$ be a (valid) partition of $V$ that solves $MSMTP$ for $G'$. Note that none of $V_1, \ldots, V_k$ are empty, otherwise we could move a single vertex to the empty set and it would have a lower $SMTP$ value for this graph. Therefore the total number of edges over all spanning trees of $G'(V_1), \ldots, G'(V_k)$ is $\sum_{i=1}^{k} |V_i| - 1 = |V| - k$. I claim that a k-coloring of $G$ exists if and only if $SMTP_{G'}^k(V_1, \ldots, V_k) = |V| - k$.

If $SMTP_{G'}^k(V_1, \ldots, V_k) = |V| - k$, then the maximum spanning tree for each subset only contains edges with weight 1. In particular, this means that no subgraph $G'(V_i)$ has an edge with weight 2, so there are no edges between any vertices of $V_i$ in $G$, otherwise the weight 2 edge would be included in the max spanning tree. Thus the partition $V_1, \ldots, V_k$ gives a $k-$coloring of $G$.

For the other implication, clearly if a $k-$coloring exists for $G$, then we can find a partition $V_1, \ldots, V_k$ such that $G'(V_i)$ only has edges of weight 1 between vertices. Then $SMTP_{G'}^k(V_1, \ldots, V_k) = |V| - k$ follows.

Therefore any algorithm which solves MSMTP also decides if $G$ has a k-coloring, which is NP-Complete for $k \geq 3$.

## B Iterative refinement algorithm

We present the pseudocode for the iterative UPEM maximization procedure that follows local clustering in Algorithm 2. In lines 4-15, we check how swapping vertices between partitions affects the overall UPEM score and take a fraction of the best swaps in line 15. We then execute the swaps and check if the UPEM score has increased. If it has not, we terminate the algorithm, otherwise we continue until $n$ iterations has passed. We take $n = 10$ in practice, and note that almost always the algorithm terminates before 10 iterations pass.

## C Correcting erroneous haplotype blocks by filling

After obtaining a set of partitions $P_1, \ldots, P_l$ of subsets of reads $B_1, \ldots, B_l$ according to the local haplotyping procedure described in Section 2.3, we correct poor clusters by the following procedure. After computing the UPEM scores for

**Algorithm 2:** Iterative refinement of UPEM score

```
Input  : Partition P = {S_1, ..., S_k}, iterations n
Output: A modified optimized partition P = {S'_1, ..., S'_k}
1  for i=1 to n do
2      OldScore ← UPEM(P)
3      L ← ∅
4      S ← ⋃_{i=1}^{k} S_i
5      for S_i in P do
6          for r in S_i do
7              for S_j in P, S_j ≠ S_i do
8                  Compute change in UPEM, Δ(r, S_j) by moving r from S_i to
                     S_j
9                  if Δ(r, S_j) > 0 then
10                     L ← L ∪ {(r, S_i, S_j, Δ(r, S_j))}.
11                 end
12             end
13         end
14     end
15     L ← reverse sort L by L[j] → Δ(r, S_j)
16     for k = 1 to ⌈|S|/n⌉) do
17         (r, S_i, S_j, Δ(r, S_j)) = L[k]
18         S_j ← S_j ∪ {r}
19         S_i ← S_i \ {r}
20     end
21     NewScore ← UPEM(P)
22     if NewScore < OldScore then
23         Reverse the moves made in Lines 16-20
24         Return P
25     end
26 end
27 Return P
```

every partition, we use a simple 3.0 IQR (inter-quartile range) outlier detection method for the distribution of UPEM scores. For an outlying partition $P_i$ of the read set $B_i$, if a partition $P_{i-1}$ is not an outlier, we remove $P_i$ and extend $P_{i-1}$ to include $B_i$. To do this, we run a subroutine of Algorithm 1 where we skip the clique finding procedure and instead treat the partition $P_{i-1}$ as the initial clusters. We then run lines 9-12 of Algorithm 1 where in line 9 we iterate over $v \in B_i \setminus B_{i-1}$ instead.

## D Genotype polishing using VCF

We constrain the haplotypes using the VCF as follows. For every variant indexed over $1 \leq i \leq m$, let $c(i, j, a) \in \mathbb{R}$ be a value representing the confidence of

calling allele $a$ at index $i$ for the haplotype represented by $P[j]$. We produce $k-$haplotypes according to Algorithm 3.

**Algorithm 3:** Polishing output haplotypes using genotype information.

```
Input  : Partition P, Genotyping information
Output: k = |P| haplotypes H_1, ..., H_k ∈ {0,1,2,3,−}^m
Initialize H_1, ..., H_k, H_i[n] = − for all 1 ≤ n ≤ m
for i = 1 to m do
    for j = 1 to k do
        for a ∈ {0,1,2,3} do
            L ← c(i,j,a)
        end
    end
    L ← reverse-sort L
    for c(i,j,a) ∈ L do
        if |{H_n : H_n[i] = a}| < # of a in VCF file and H_j[i] = − then
            H_j[i] = a
        end
    end
end
```

For the function $c(i, j, a)$ describing the confidence for calling allele $a$ at position $m$ for haplotype $j$, we choose the function

$$c(i, j, a) = \frac{|\{r \in P[j] : r[i] = a\}|}{|\{r \in P[j] : r[i] \neq a\}| + 1}.$$

Note that H-PoP uses the same method for polishing, but they choose a confidence function that is a difference instead of a ratio. A ratio does a better job because for a particular haplotype if there are 100 reads that have allele 1 at position 1, and 50 reads that have allele 2, we believe that this is a less confident call than a haplotype with 50 reads with allele 1 and 0 reads that have other alleles.

## E MSMTP vs MEC

We ran flopp on four different simulated datasets (see Section 3.2) and calculated the SMTP score with the weight function $w(r_1, r_2) = d(r_1, r_2)$, see Equation 1, and the MEC score for each local partition.

We varied the coverage between 10x to 20x for a simulated 4x ploidy genome. We also varied the length of the local partition blocks by manually changing the parameter $b$ mentioned at the end of Section 2.3 over three different values for each different data set to investigate how the size of the local clusters affects the SMTP and MEC relationship. The results are shown in Figure 6

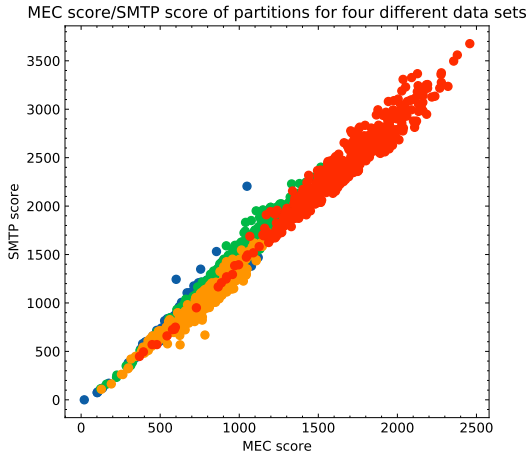


**Fig. 6.** A plot of the SMTP vs MEC score for every local cluster generated by flopp over four different simulated data sets across different coverages with each color representing a distinct data set.

## F WHP Results on 3x ploidy data

We ran WhatsHap Polyphase (WHP) on the simulated genomes as described in Section 3.2. We ran WHP with default settings. In particular, the block-cut sensitivity setting which determines the contiguity of the blocks was set at the default value of 4. There were around 67000 variants in the contig of 3.02 Mb; The N50 over all iterations, coverages, and read types of WHP's output never exceeded 20. The results are shown in Figure 7.

We found that WHP takes a conservative approach as $> 15\%$ of the variants were not called by WHP, even on 20x coverage per haplotype data. This is included as part of the Hamming error rate in our tests. In [1], the authors also identify that WHP tends to be extremely conservative with respect to cutting haplotype blocks and calling variants. One of the implications is that the Hamming error rate for WHP on our data set is significantly higher than that reported in their study [33]. However, there are several differences in the settings for our respective studies, which we hypothesize may contribute to that disparity. For example, there are relatively high rates of heterozygosity in our test data set (45 bp between SNPs on average), which is less the case on the reported test results for WHP [33], as they test on artificial human polyploid chromosomes created by combining different human chromosomes; it is well known that human chromosomes have much lower rates of heterozygosity than potato genomes which have variants on average $<$ 50bp apart [35].

Another difference is in the simulator we used to generate long-read data for our benchmarks (PaSS [39], NanoSim[37]), which may differ in error profile than the data in their experiments.

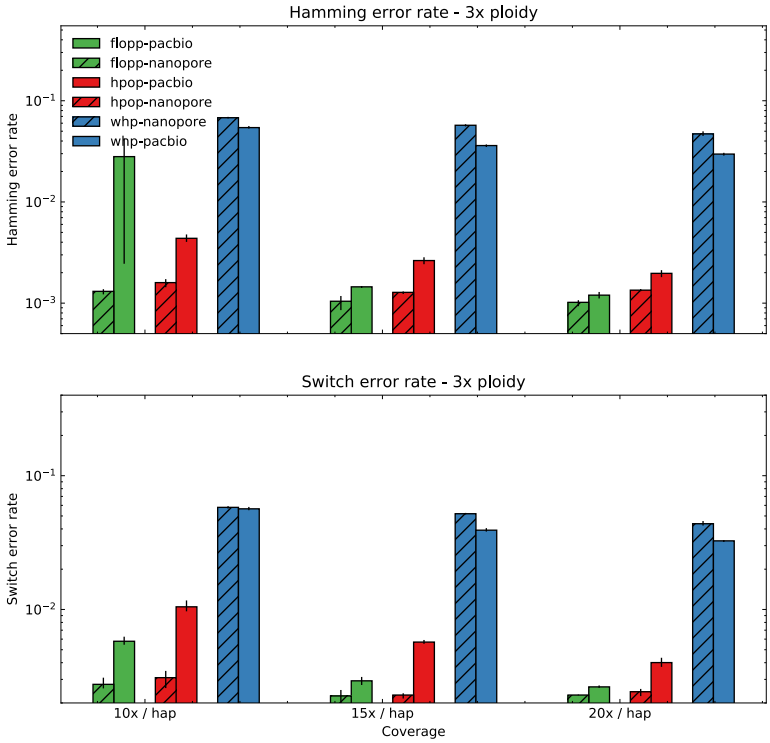


**Fig. 7.** SWER and Hamming error rates of flopp, H-PoPG, and WHP on 3x ploidy data set. The generation of the data set is described in Section 3.2. The error bars represent the lowest and highest values for the metric over three iterations. The Hamming and switch error rates are averaged over all broken blocks for WHP.

Finally, we noticed that on polyploid genomes that have very similar haplotypes, i.e. genomes which exhibit heavy amounts of collapsing, the run time of WHP is much faster than on our simulated data set. Regardless, we believe that more testing is needed to identify the regimes in which the various types of long-read based phasing algorithms perform optimally.

## G Results on simulated data sets with different rates of heterozygosity

We include two additional tests based on the simulated data set described in the paper under the same testing conditions with the rates of heterozygosity changed from 45 bp to 90 bp and 135 bp between SNPs on average (mean). The results are shown in Figure 8 and Figure 9. At higher rates of heterozygosity, flopp has a higher likelihood of a major switch error occurring which may result

in a high hamming error rate. However, the switch error rate for flopp is still better than H-PoP, showing that at on a local scale, flopp still performs well.

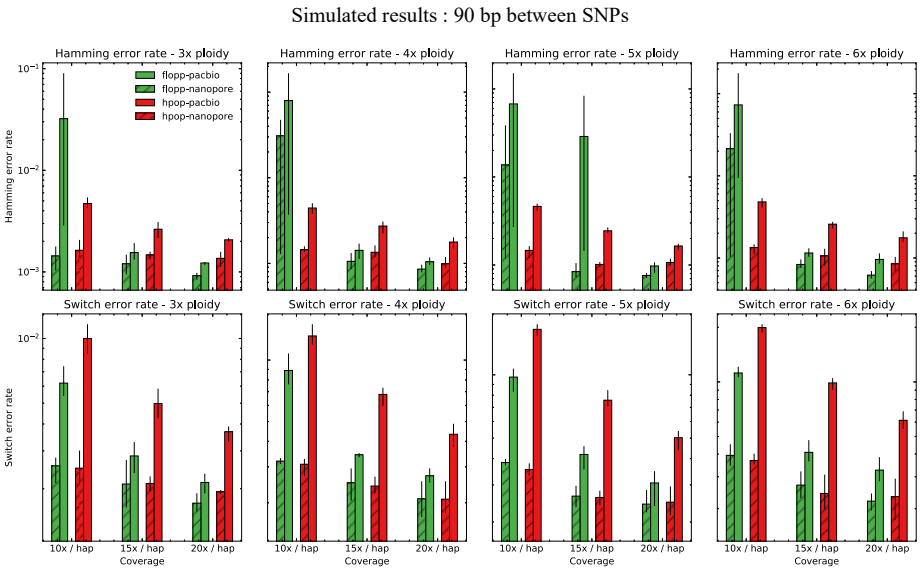


**Fig. 8.** The mean switch error rate and Hamming error rate from testing on simulated data sets as described in Section 3.2 except with 90 bp between SNPs on average (instead of 45bp). The error bars represent the lowest and highest values for the metric over three iterations.

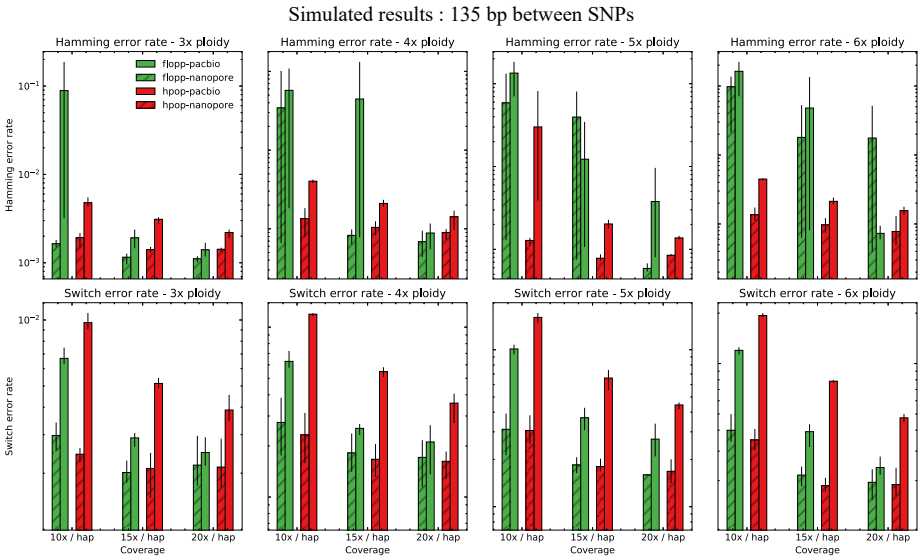


**Fig. 9.** The mean switch error rate and Hamming error rate from testing on simulated data sets as described in Section 3.2 except with 135 bp between SNPs on average (instead of 45bp). The error bars represent the lowest and highest values for the metric over three iterations.