HAT

haplotype assembly tool using short and error-prone long reads

Shirali Hossein Zade, Ramin; Urhan, Aysun; Assis de Souza, Alvaro; Singh, Akash; Abeel, Thomas

DOI
10.1093/bioinformatics/btac702

Publication date
2022

Document Version
Final published version

Published in
Bioinformatics (Oxford, England)

Bioinformatics, 38(24), 2022, 5352–5359
https://doi.org/10.1093/bioinformatics/btac702
Advance Access Publication Date: 29 October 2022
Original Paper

Genome analysis

HAT: haplotype assembly tool using short and error-prone long reads

Ramin Shirali Hossein Zade [1], Aysun Urhan[1,2], Alvaro Assis de Souza[1], Akash Singh[1] and Thomas Abeel [1,2,*]

[1]Delft Bioinformatics Lab, Delft University of Technology Van Mourik, 2628 XE Delft, The Netherlands and [2]Infectious Disease and Microbiome Program, Broad Institute of MIT and Harvard, Cambridge, MA 02142, USA

*To whom correspondence should be addressed.

Associate Editor: Can Alkan

Received on July 20, 2022; revised on September 16, 2022; editorial decision on October 21, 2022; accepted on October 25, 2022

Abstract

Motivation: Haplotypes are the set of alleles co-occurring on a single chromosome and inherited together to the next generation. Because a monoploid reference genome loses this co-occurrence information, it has limited use in associating phenotypes with allelic combinations of genotypes. Therefore, methods to reconstruct the complete haplotypes from DNA sequencing data are crucial. Recently, several attempts have been made at haplotype reconstructions, but significant limitations remain. High-quality continuous haplotypes cannot be created reliably, particularly when there are few differences between the homologous chromosomes.

Results: Here, we introduce HAT, a haplotype assembly tool that exploits short and long reads along with a reference genome to reconstruct haplotypes. HAT tries to take advantage of the accuracy of short reads and the length of the long reads to reconstruct haplotypes. We tested HAT on the aneuploid yeast strain *Saccharomyces pastorianus* CBS1483 and multiple simulated polyploid datasets of the same strain, showing that it outperforms existing tools.

Availability and implementation: https://github.com/AbeelLab/hat/.

Contact: t.abeel@tudelft.nl

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Most eukaryotes have more than one copy of each chromosome, and some species have more than two homologous copies of each chromosome (i.e. polyploids), which is common in plants (Ramsey and Schemske, 1998). In genetics, a haplotype is the combination of individual alleles (one allele of each gene) located on the same chromosome. Because these alleles are located on the same chromosome, they are passed on together to the next generation (Crawford and Nickerson, 2005). Haplotype assembly or reconstruction refers to the task of reassembling each individual haplotype. The need to reconstruct haplotypes arises from the inability of current DNA sequencing technologies, such as next-generation (NGS) and third-generation (TGS) sequencing, to read a chromosome's sequence from beginning to end. These technologies instead sequence shorter fragments called reads. In addition, chromosome separation before sequencing requires complicated and expensive lab work that is not feasible for most studies. Therefore, it is more common to sequence chromosomes together, and then use computational methods to separate the reads and reconstruct the haplotypes. A monoploid reference genome consists of a mosaic structure of haplotypes with allelic combinations that do not co-occur within any haplotype. Additionally, some of the alleles found in the haplotypes are absent in the monoploid reference. In contrast, with a haplotype-resolved reference, we can understand genetic variation and link phenotypic traits with the associated alleles in the haplotype better.

It is significantly more challenging to reconstruct haplotypes for polyploid genomes than for diploid genomes. If one of the haplotypes of a diploid genome has been phased (i.e. the said haplotype has been inferred), it is trivial to determine the alleles of the other haplotype based on this. On the other hand, in polyploid genomes, other haplotypes may have the same or different alleles (Garg, 2021). Hence, the phasing of one haplotype does not clearly indicate what alleles are present in other haplotypes.

Recognizing the widespread use of NGS and TGS, it is imperative to develop algorithms for polyploid haplotype reconstruction from sequencing reads to facilitate various research applications such as finding compound mutations that cause a disease (Ng *et al.*, 2009), or studying yield-related markers that are located in a haplotype that can be used in plant breeding programs (Bhat *et al.*, 2021).

In the past years, few tools have tackled this problem. nPhase (Abou Saada *et al.*, 2021) and Whatshap (Schrinner *et al.*, 2020) are among some examples of recently developed tools.

This study presents HAT, a haplotype assembly tool that combines short reads and error-prone long reads along with a reference genome to reconstruct haplotypes. Similar to Ranbow (Moeinzadeh *et al.*, 2020), HAT first creates seeds from short reads, but then it expands the seeds with long reads. We benchmark HAT against Whatshap and nPhase because both use long reads to phase haplotypes. Using simulated and real yeast genome data, we demonstrate that HAT outperforms both Whatshap polyphase and nPhase in terms of contiguity and the accuracy of phased alleles.

2 Materials and methods

2.1 Data

Using both simulated and real data is essential to test HAT properly. Simulated data provide the ground truth of haplotypes to evaluate phasing accuracy, and the real data validate HAT's performance.

2.1.1 Simulated data

We use Haplogenerator (Motazedi *et al.*, 2018) to simulate haplotypes from the base genome—chromosome ScII of *Saccharomyces pastorianus* CBS1483 with accession number ASM1102231v1 (ChrSc2) (Salazar *et al.*, 2019). The ground truth is the simulated haplotypes, and ChrSc2 base sequence is the reference. Next, we simulate reads with 20× coverage per haplotype, similar to the simulation design used in previous studies including nPhase. We use Badread (Wick, 2019) version 0.2.0 and ART (Huang *et al.*, 2012) version 2.5.8 to simulate reads similar to Oxford Nanopore Technology (ONT) and to Illumina's HiSeq 2500, respectively. Badread is used with default parameters, ART parameters are available in Supplementary Table S2. Supplementary Table S4 shows the simulated ONT reads' error rates and compares them to the real data. In total, six datasets are generated for ploidy levels 3, 4 and 5, with low and high heterozygosity. For the low heterozygosity datasets, we set the parameters of Haplogenerator to produce the same number of SNPs/Insertions/Deletions as the chromosomes ScII, SeIII-ScIII and ScVIII of CBS1483 which are triploid, tetraploid and pentaploid, respectively. Because chromosomes SeIII-ScIII and ScVIII are smaller than ScII, the chromosome we use for the simulations, we multiply the number of SNPs/Insertions/Deletions by the ratio of genome sizes. For the high-heterozygosity datasets, we fit a lognormal distribution on the distances (Motazedi *et al.*, 2018) between consecutive SNPs/Insertions/Deletions of chromosome ScII, SeIII-ScIII and ScVIII and use the parameters on Haplogenerator. The parameter settings for Haplogenerator are in Supplementary Table S1.

Then, the short reads are mapped to the monoploid reference genome using BWA-MEM (https://arxiv.org/abs/1303.3997) with default parameters. We obtain variations using FreeBayes (https://arxiv.org/abs/1207.3907) version 0.9.21 from the short-read alignments. We use vcffilter from vcflib (https://www.biorxiv.org/content/10.1101/2021.05.21.445151v1) package version 1.0.2 to extract the SNPs. We map the long reads using minimap2 (Li, 2018) version 2.13-r858-dirty. Parameters for all tools are in Supplementary Table S2. The short- and long-read mapping, ploidy of the chromosome and the SNPs are the inputs for HAT.

2.1.2 Real data

We reconstruct the haplotypes of CBS1483, which is aneuploid and has ploidy ranging from one to five. It consists of ONT and paired-end Illumina reads that are available under the BioProject PRJNA522669. There are four ONT runs in this BioProject, and we used all of them in this study. Short reads have coverage of 159× and are 151 bp, Nanopore reads have coverage of 72× with an average read length of 7 kb and N50 of 10 kb. We use the ASM1102231v1 assembly as the reference genome of CBS1483.

Moreover, we reconstruct the haplotypes of *Brettanomyces bruxellensis* strain GB54, a triploid genome which has higher heterozygosity, and longer chromosomes than CBS1483. The longest chromosome of GB54 is 4 Mb which is three times larger than CBS1483 largest chromosome. The ONT and paired-end Illumina reads are available under the BioProject PRJEB40511. The Illumina short reads are 75 bp and 30× coverage. The nanopore long reads have the average read length of 12 kb, 82× coverage and 23 kb N50. We use the DEBR_UMY321v1 assembly as the reference genome of GB54.

In both real datasets, the SNPs and the alignments are obtained with the same method as in the simulated datasets.

2.2 HAT method

HAT reconstructs haplotypes by linking alleles at SNP loci together using short and long reads. HAT comprises three components—initialization, iteration and assembly. Initialization creates the first phased blocks. The iteration expands the phased blocks and finds alleles of all haplotypes. Then, HAT clusters the reads and assembles haplotypes using these clustered reads. An overview of the HAT algorithm can be seen in Figure 1.

2.2.1 Initialization

In initialization (see Fig. 1A), the multiplicity blocks are found, and then the first phased blocks are created. Phased blocks are a set of consecutive SNP loci in the phase matrix where the alleles are connected. First HAT creates seeds, which are a combination of consecutive SNP loci covered by the same short read; a single seed can be as small as two SNP loci. To create the seeds, we determine the SNP loci each short read is covering. If a read covers more than two SNP loci, we create all combinations of consecutive SNPs with different lengths and starting points. When we create a seed, based on the alleles present in the short reads that cover it, we obtain a set of combinations of alleles. While processing a short read, if the seed it would create already exists, no new seed is created and only the combination of alleles in the new read is added to the existing seed. In addition, we store the number of reads supporting each combination of alleles. Next, we filter the combination of alleles and the seeds. The combinations of alleles with fewer than five reads supporting are removed to remove erroneous combinations of alleles. If a seed ends up with fewer than two combinations of alleles, it is removed.

Next, HAT finds overlapping seeds and keeps only one of them because each SNP locus should be at most in one of the seeds to avoid conflicts. When HAT finds overlapping seeds, we check the number of combinations of alleles in each seed, the support of the seeds and the first SNP locus of the seeds, then HAT picks the seed

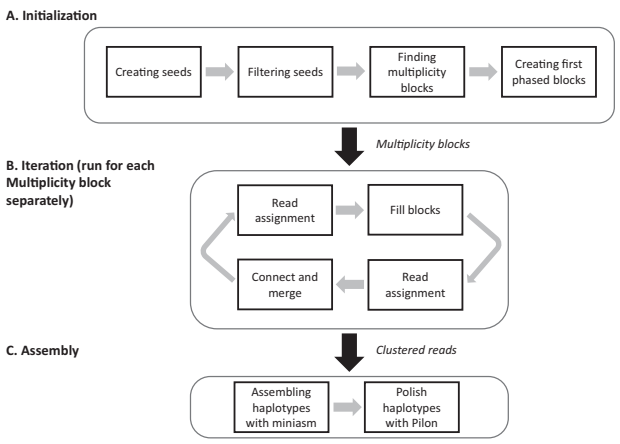

Fig. 1. HAT overview. (**A**) HAT creates seeds based on short-read alignments and the location of SNPs. Then, it removes the combinations of alleles with low support as well as overlapping seeds. Next, HAT finds multiplicity blocks and creates the first phased blocks within them. (**B**) HAT assigns reads to the blocks and haplotypes; based on these read assignments, it fills the unphased SNPs within blocks. (**C**) Finally, HAT can also use miniasm to assemble haplotype sequences for each block and polishes the assemblies using Pilon, but this step is optional

with the maximum number of combinations of alleles. If there are two overlapping seeds with the same number of combinations, we pick the longest one.Then, HAT detects the regions that contain at least two different haplotypes, which we call multiplicity blocks. We use the sorted set of seeds as the input for Algorithm 1 to find multiplicity blocks and their corresponding multiplicity. The seeds are sorted with respect to the number of combinations of alleles, and when two seeds have the same number, the one with an earlier position of the first SNP locus will come first. Once Algorithm 1 has identified the multiplicity blocks if the estimated multiplicity of the block exceeds the ploidy of the chromosome, it is decreased to the ploidy of the chromosome. In such cases, HAT eliminates combinations of alleles with low support until each seed has the same number of combinations as the chromosome's ploidy. HAT creates a separate phase matrix for each multiplicity block. Each row of the phase matrix corresponds to one of the haplotypes, and each column is representative of an SNP locus within the multiplicity block.

Algorithm 1: Find multiplicity blocks algorithm. The find multiplicity blocks algorithm takes the sorted set of seeds and a distance_parameter as input, and it returns multiplicity blocks as output. The variable *Multiplicity_blocks* is a dictionary that with multiplicity blocks as keys and multiplicity of the region as values. Each multiplicity block has a start and end position as well as an estimated multiplicity for that region. The *Alleles* function in the algorithm gets a seed as input and returns the number of combinations of alleles the seed has. The *Multiplicity* function in the algorithm gets a list of seeds as input and returns the highest number of combinations of alleles within them.

```
Parameters: : d ;          /* distance_parameter */
Inputs:     : S ;          /* sorted set of seeds */
Returns:    : MB ;         /* Multiplicity blocks */
FindMultiplicityBlocks (S, d)
    cores ← [ ];
    foreach seed ∈ S do
        if length(cores) = 0 then
            new_cores ← [seed];
            cores.append(new_cores);
            continue;
        closest_core ← The closest core to the seed;
        if Distance(seed, closest_core) ≤ d then
            if Alleles(seed) = Multiplicity(closest_core) then
                cores[closest_core].add(seed);
        else
            new_cores ← [seed];
            cores.add(new_cores);
    Multiplicity_blocks ← [[ ] ∗ length(cores)];
    foreach seed ∈ S do
        closest_core ← The closest core to the seed;
        Multiplicity_blocks[closest_core] ← seed;
    MB ← {};
    foreach m_b ∈ Multiplicity_blocks do
        l ← Most left position of m_b;
        r ← Most right position of m_b;
        m ← Maximum multiplicityof m_b;
        MB[(l, r)] ← m;
    return MB;
```

Finally, HAT generates the first phased blocks. First, HAT removes the seeds with fewer combinations of alleles than the estimated multiplicity of the block. Next, we use the combinations of alleles of the seeds to fill the phase matrix in the columns the seed is covering. Each seed creates a separate phased block because the relation of combinations of alleles of different seeds is unclear to one another. The SNP loci that do not belong to any block are added to the closest block to them.

2.2.2 Iteration

The iterative part of HAT (Fig. 1B and Supplementary Fig. S1B) continues until there is only one block and the phase matrix is full, or if the blocks and the phase matrix stop updating. We run the first iteration with short reads and the rest with long reads.

An essential step of the iterative HAT algorithm is assigning short and long reads to haplotypes in blocks. Each stage of the iterative part uses these assigned reads. Therefore, after both *Fill blocks* and *Connect and merge* steps, reads are reassigned to the haplotype blocks based on the latest changes.

First, for every read, we check the phased SNP loci that it covers within a block. If the combinations of alleles at those loci are unique for each haplotype, the read is assigned to the block. Then, the alleles of the read located at the phased SNP loci the read is covering within the block are compared with the alleles of each row of the phase matrix. The read is assigned to the haplotype if the Hamming distance to the row is less than hamming_parameter, which changes with each run of the algorithm. When assigning long reads to the haplotypes, the hamming_parameter is small (1) to accommodate sequencing errors. As the phasing algorithm proceeds, we increase hamming_parameter to 3 to be less strict with the assignments, because there are more shared phased SNP locus within the block and the read.

The next stage of the iterative component is connecting and merging consecutive phased blocks. To connect two blocks, we use reads assigned to both blocks. We iterate over the blocks based on their location. If there is a one-to-one connection between all the haplotypes of two blocks with enough support, the blocks are merged, and the rows of the second block are switched so that the connected haplotypes of the first and second haplotypes are in the same row. Two haplotypes are connected if the number of reads supporting the connection is more than 1 in the first and second iterations, and 3 in the rest.

In the blocks' filling step, we use all reads assigned to haplotypes of a block as input and process them to find the allele of unphased SNPs within the block by a majority voting between the reads of the haplotype. If the number of the reads supporting the majority vote allele is greater than 2, the allele is assigned to the haplotype's SNP locus, and that cell of the phase matrix is filled. This phase might lead to some SNP loci being phased in some haplotypes but not in others. When iteration converges, HAT assigns long and short reads to haplotypes of each phased block using the read assignment module.

2.2.3 Assembly

Optionally, HAT can assemble the reads to reconstruct sequence of the haplotypes using miniasm (Vaser *et al.*, 2017) version 0.3-r179 and the clustered long reads, then polish the assemblies using Pilon (Walker *et al.*, 2014) version 1.24 and the clustered short reads. This part of HAT is optional. HAT uses miniasm and Pilon with default parameters. Users can use HAT to only cluster the reads and create the phase matrix and then use a tool of their choice to reconstruct the sequence of the haplotypes.

2.3 Output

HAT outputs the following files:

- A multiplicity block figure which illustrates the multiplicity blocks and their level over the chromosome.

- The clustered reads files which contain the IDs of clustered reads for the haplotypes of each phased block.
- The phase matrix file which lists the alleles of haplotypes within each phased block.
- The haplotype sequences within each phased block. This output is optional, and it is produced only if the user also requests assembly.

2.4 Evaluating HAT

We run HAT version 0.1.7, nPhase version 1.1.10 and Whatshap polyphase version 0.19.dev161+g7660dcf from the polyploid-haplotag branch on the simulated datasets and compare the phasing and read clustering accuracy. Unlike HAT and nPhase, Whatshap polyphase does not cluster the reads by default and after phasing with Whatshap polyphase, we use Whatshap haplotag to cluster the reads for evaluation purposes. The parameters of Whatshap haplotag are mentioned in Supplementary Table S2. To calculate the phasing accuracy of HAT, first we find a one-to-one mapping between the haplotypes HAT identifies within each block and the real haplotypes. Then, we compare the allele of each haplotype at the SNP loci from the phase matrix to the ground truth. We count the number of correct SNPs for all the blocks and haplotypes and calculate accuracy as the count of correct SNPs divided by the total number of SNPs within the multiplicity blocks. To calculate the phasing accuracy of nPhase and Whatshap, we first find the haplotype which is the most similar to each cluster based on the cluster's and haplotype's alleles at the SNP loci the cluster covers. Then, we divide the count of correctly phased SNPs by the total number of SNPs within the clusters.

Then, we assess the accuracy of read clustering. Since the simulated reads are already labeled with their native haplotype, we calculate the clustering accuracy by counting the number of reads clustered correctly. We also count the number of phased blocks to evaluate the reconstructed haplotypes' completeness.

In addition to simulated data, we investigate the haplotypes HAT creates for the real CBS1483 and GB54 data.

3 Results

3.1 Conceptual overview of HAT using the example of a triploid chromosome

To provide an overview of the HAT algorithm, we consider the triploid chromosome ScII of *S.pastorianus* CBS1483 (ChrSc2), for a step-by-step discussion of HAT. The HAT algorithm consists of three main steps: (i) initialization, (ii) iteration and (iii) assembly. The input is a combination of both short and long reads, along with a reference genome. HAT will produce read clusters per haplotype when run in default settings. If the optional assembly parameter is supplied by the user, HAT will also generate the haplotype sequences.

In the initialization step, HAT builds prototype phased blocks from seeds within multiplicity blocks. Phased blocks are fully resolved haplotype segments while multiplicity blocks are genomic regions presenting sufficient variants for phasing and have an estimated ploidy associated. The initialization consists of three steps. First, HAT uses the alignment of short reads to the reference to find well-supported combinations of variant alleles, called seeds. In our example of ChrSc2, 528 SNPs were used to create 25 335 combinations, which are filtered down to 119 by removing the combinations with low support (see Section 2). Next, HAT constructs multiplicity blocks from the seeds with Algorithm 1. Finally, HAT uses seeds with a matching number of combinations of alleles to create the first phased blocks within each multiplicity block. In the example of ChrSc2, HAT found 16 multiplicity blocks (see Supplementary Fig. S3).

During the iterative phase, HAT processes each multiplicity block to phase the remaining SNPs and create bigger phased blocks within a multiplicity block. It consists of two sections: (i) filling blocks and (ii) merging blocks. Before running each section, HAT assigns reads to blocks and haplotypes based on the SNPs each read covers and their similarity to the phased SNPs. Supplementary Table S3 shows how each step of the iterative algorithm improves the phasing of ChrSc2. The iteration stops when there is no improvement over the previous step. The first iteration uses the short reads while the remaining iterations use long reads. In our experiments with real and simulated data, HAT converges in <4 iterations. Increasing the number of iterations for the short reads does not change the overall phasing performance because the blocks are bigger than the linking range of the short reads.

Upon convergence, there are 23 phased blocks and only 23 unphased alleles from the SNP loci within the multiplicity blocks. We use miniasm on the long reads assigned to haplotypes of each phased block to assemble them. Then, we polish the assemblies with the short reads assigned to haplotypes using Pilon.

3.2 HAT outperforms state-of-the-art on simulated data

To evaluate HAT, we use simulated datasets, consisting of short and long reads, and alignments to the haplotypes. Details of the simulation are described in Section 2. Summary statistics of the simulated datasets are reviewed in Table 1.

We compare HAT to nPhase and Whatshap polyphase using the various metrics (see Section 2); Table 2 summarizes the performance of the tools on the simulated datasets. First, we compare long-read clustering accuracy. The number of long reads clustered incorrectly by HAT is lower than that of both nPhase and Whatshap for all ploidy levels: HAT's error rate ranges from <1% (triploid high heterozygosity) to 24% (pentaploid low heterozygosity), whereas for nPhase, the range is 5% (tetraploid high heterozygosity) to 38% (pentaploid low heterozygosity) and for Whatshap, it is 7% (tetraploid high heterozygosity) to 22% (triploid high heterozygosity).

For all datasets, HAT successfully phases at least 90% of the SNPs, and the accuracy is the highest at 98% for the triploid high heterozygous genome (last column in Table 2). In all datasets, Whatshap has the lowest accuracy and HAT has the highest. Note that the phasing accuracy of HAT is calculated only for the SNPs inside the multiplicity blocks, but the multiplicity blocks cover almost all of the SNPs on the chromosome, with the lowest coverage being 78% for the pentaploid low heterozygous genome (Table 3). Similarly, for nPhase and Whatshap, we calculated the phasing accuracy based on the clusters each tool generates.

As shown in Table 2, HAT phases fewer total SNPs than nPhase and Whatshap. That is because HAT does not attempt to phase the areas far from the seeds. A few reads cover both the core of the

Table 1. Descriptive statistics of the simulated datasets and ChrSc2, the base chromosome used for simulations

Dataset	Ploidy	Simulated SNPs	No. of SNPs found by Freebayes	No. of short reads	No. of long reads
Triploid low heterozygosity	3	1230	687	194 910	3295
Triploid high heterozygosity	3	6398	4143	194 910	3441
Tetraploid low heterozygosity	4	1144	506	259 880	4423
Tetraploid high heterozygosity	4	12 072	7512	259 880	4358
Pentaploid low heterozygosity	5	1606	504	324 850	5433
Pentaploid high heterozygosity	5	17 802	7232	324 850	5394
CBS1483 chromosome ScII	3	—	528	428 802	8051

Table 2. HAT outperforms nPhase and Whatshap in phasing accuracy on simulated data

Dataset	Tool	Read clustering error		Total reads phased		SNP phasing performance			Accuracy[b] (%)
		Short[a]	Long	Short[a]	Long	Correct	Incorrect	Unphased	
Triploid low	HAT	14	65	5400	2122	1813	20	35	98
	nPhase	—	307	—	1268	1936	379	0	84
	Whatshap	—	225	—	1943	759	1215	0	61
Triploid high	HAT	55	13	37 291	2138	11 895	185	19	98
	nPhase	—	218	—	2829	12 701	1464	0	90
	Whatshap	—	680	—	3060	7106	4663	0	60
Tetraploid low	HAT	135	187	2466	1580	1549	34	68	94
	nPhase	—	297	—	1439	2023	335	0	86
	Whatshap	—	254	—	2229	1434	538	0	72
Tetraploid high	HAT	62	32	17 968	2252	29 039	493	37	98
	nPhase	—	219	—	4053	28 980	1990	0	94
	Whatshap	—	287	—	4142	20 550	7942	0	72
Pentaploid low	HAT	545	518	2350	2141	1341	51	74	91
	nPhase	—	662	—	1726	1714	287	0	86
	Whatshap	—	348	—	2396	1803	547	0	76
Pentaploid high	HAT	1041	266	9360	6804	31 980	1479	224	95
	nPhase	—	449	—	5264	35 450	2676	0	93
	Whatshap	—	708	—	5246	27 929	7301	0	79

Note: The first two columns show number the number of reads clustered and phased, third column lists the number of SNPs within multiplicity blocks that were phased correctly/incorrectly and unphased. The final column is the SNP phasing accuracy.

[a]The short read cluster error was calculated only for HAT because nPhase and Whatshap are designed specifically to cluster the long reads.

[b]Accuracy is defined in Section 2.

Table 3. The percentage of SNPs that are inside multiplicity blocks

Dataset	Percentage of SNPs inside multiplicity blocks
Triploid low heterozygosity	92
Triploid high heterozygosity	99
Tetraploid low heterozygosity	87
Tetraploid high heterozygosity	99
Pentaploid low heterozygosity	78
Pentaploid high heterozygosity	98

multiplicity block and these regions that are far, making phasing of these regions less reliable. That means the result of HAT can be incomplete, but it has higher accuracy because it only works in reliable regions.

To assess the phasing contiguity, we checked the number of phased blocks in the HAT output and report that for highly heterozygous cases HAT can phase almost all of the haplotypes. HAT creates two phased blocks for the triploid case and three phased blocks for the tetraploid one. For cases with low heterozygosity, HAT creates 15, 30 and 23 phased blocks for the triploid, tetraploid and pentaploid genomes. This is expected because these genomes are largely identical, and it is not possible to connect the phased blocks. In contrast, we also note that for the highly heterozygous pentaploid dataset, HAT creates 33 phased blocks although it has 96% phasing accuracy, an outcome likely caused by the high ploidy level.

3.3 HAT shows robust performance on real data

Since there are not many chromosome-level polyploid assemblies available, the disparity between simulated and real genomes can be significant. Hence, we are evaluating HAT on the real *S.pastorianus* CBS1483 dataset to corroborate the results from the simulated datasets as well. CBS1483 is a valid test model because it is aneuploid and has various ploidies ranging from one to five. Additionally, the chromosomes are small and easy to investigate. We report read clustering and phasing results for seven chromosomes of CBS1483 representing various levels of ploidy, heterozygosity and length in

Table 4. HAT results on CBS1483 real data

Chromosome name	Ploidy	No. of SNPs	% phased regions	Alleles within blocks	
				Phased	Unphased
ScI	3	619	54	1384	60
ScII	3	528	14	922	23
ScIV	3	1643	20	3424	168
ScIX	2	195	10	335	43
ScVIII	5	417	14	687	4
SeI	2	21	<1	8	0
SeVII-ScVII	3	341	4	444	5

Note: These chromosomes are a representative subset of all chromosomes of CBS1483.

Table 4. For the highly heterozygous chromosome ScI (see Fig. 2), multiplicity blocks that HAT finds cover 54% of the whole sequence and within these blocks HAT phased 96% of the alleles. Although ScIV contains a large number of SNPs, it is the largest chromosome, and all the SNPS are concentrated around the centromere and thus, the % of phased regions is lower. SeI, on the other hand, is one of the shortest chromosomes (185 kb long) and there are very few SNPs, meaning that the haplotypes are identical in most positions on the chromosome. For that reason, HAT phases <1% of the chromosome.

We observe the haplotype sequences created by miniasm and polished by Pilon for the multiplicity block 153738, 163604 in chromosome ScII (Fig. 3). This multiplicity block is only 8 kb long, and the estimated ploidy for that region is 2. To visually investigate the accuracy of haplotype reconstruction, we map the clustered reads to haplotypes 1 and 2 reconstructed by HAT and view the alignment in Integrative Genome Viewer (iGV). Figure 3 depicts the alignment of clustered reads of CBS1483 Chromosome ScII to the sequence of the first haplotype. The reads that belong to each haplotype have matching alleles that can differentiate them from the reads of other haplotypes. We, therefore, demonstrate that the HAT

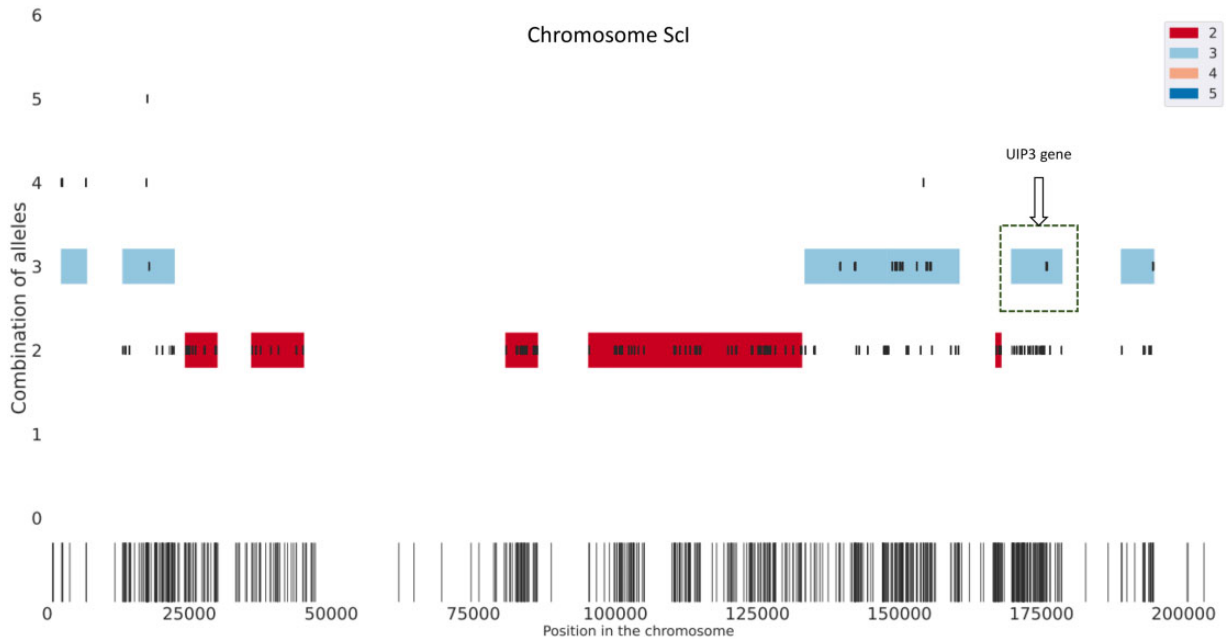

Fig. 2. Multiplicity blocks HAT finds for chromosome ScI of CBS1483. The output of finding multiplicity blocks algorithm on real data, chromosome ScI of CBS1483. The long, black vertical lines at the bottom show the SNPs and their positions on the chromosome found by FreeBayes. From these SNPS, HAT finds the seeds shown in short, black vertical lines in panel above the SNPs. The seeds are placed vertically based on the number of combination of alleles they have, ranging from one to six (y axis). HAT uses these seeds to find multiplicity blocks, which are shaded regions encapsulating the seeds and the color of the region indicates the estimated multiplicity level. See the legend for the colors corresponding to different multiplicity levels. The dashed box covers the multiplicity block that contains the UIP3 gene

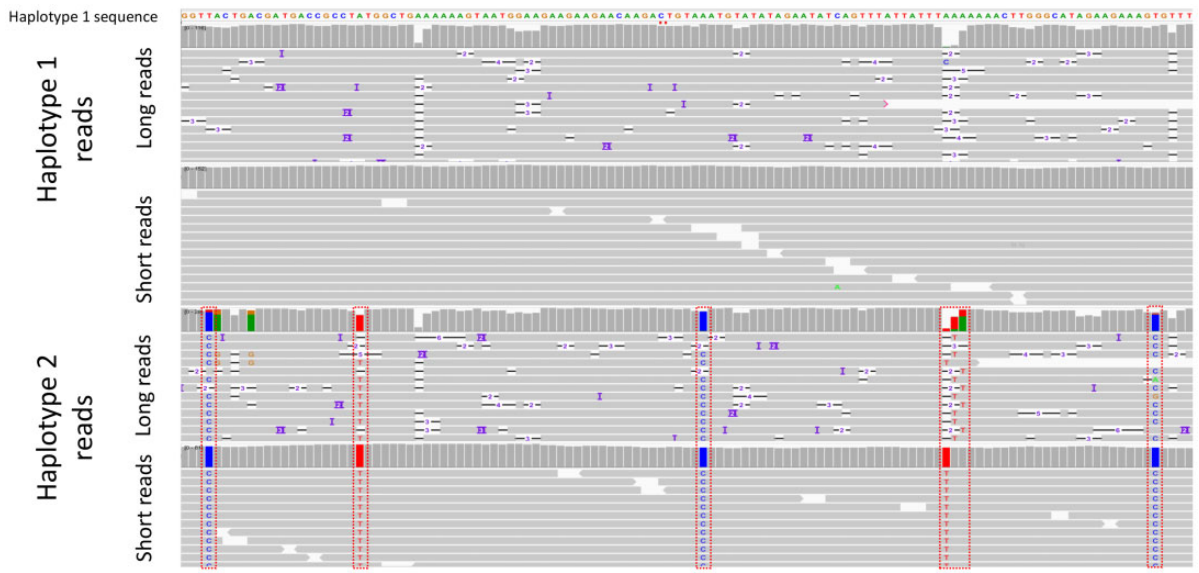

Fig. 3. HAT can accurately cluster reads to reconstructed haplotypes. We aligned short and long reads to haplotype 1 (top two rows) and haplotype 2 (bottom two rows) phased by HAT for the multiplicity block covering the positions from 153 738 to 163 604 of ChrSc2 and visualized the alignment using iGV. Haplotype 2 reads differ significantly from haplotype 1 reads at five positions (outlined with dashed rectangles)

algorithm for read clustering and finding multiplicity blocks works on real data.

Previous studies show that the UIP3 gene is removed in some of the haplotypes of CBS1483 chromosome ScI (Salazar *et al.*, 2019). We investigate the same gene in HAT output by aligning the reads HAT clustered in the multiplicity block covering the positions from 169765 to 178549 where UIP3 is located (see Fig. 2), to UIP3 sequence. As expected, only haplotype 2 reads align with the gene.

Finally, we test the performance of HAT on GB54, a triploid *B.bruxellensis* yeast strain, which Abou Saada *et al.* (2021) also use to evaluate nPhase. GB54 is an interesting test set because it has longer chromosomes and the chromosomes are more heterozygous compared to CBS1483. Table 5 shows HAT's performance in phasing GB54, and Supplementary Figure S4 illustrates the multiplicity blocks HAT finds. As expected, the percentage of phased regions is much larger than that of CBS1483 (Table 4), since GB54 is more heterozygous. Additionally, when we visualize the multiplicity blocks in GB54 we observe multiple long regions in chromosomes 2, 3 and 5 where two of the haplotypes are identical. For instance, on Chr 3 the genomic region from 909325 to 1172321, all of the seeds have only two combinations of alleles, and the average ratio of the read support for the combination of alleles of seeds at these regions is 1.7. This is in line with our expectation that two of the haplotypes are identical in this region, and we get on average near twice as

Table 5. HAT results on GB54 real data

	Chromosome	% of phased regions	Alleles within blocks	
			Phased	Unphased
Chr 1	36 628	84	96 764	2001
Chr 2	23 756	82	60 066	2586
Chr 3	18 902	86	35 728	1807
Chr 4	19 377	89	50 616	884
Chr 5	10 609	63	17 867	1673
Chr 6	15 762	72	32 877	1295
Chr 7	3581	92	10 482	216
Chr 8	1327	72	2949	305

much read coverage for one of the haplotypes as there would be if there were three different haplotypes. Moreover, when Abou Saada *et al.* phased Chr 4 using nPhase, they reported that two of the haplotypes were identical at the end region of Chr 4 since they could phase only two haplotypes. However, when we look at the same location on Chr 4, we observe two small genomic regions (from 1407542 to 1430976 and from 1533678 to 1546306) where HAT can successfully phase all three haplotypes (see Supplementary Fig. S4).

The running time of HAT depends on the size of the multiplicity blocks. The bigger the multiplicity blocks are, the more reads are assigned to them. The computational complexity of running HAT is currently $O(SKn^2)$, where S is the number of SNP loci, K is the ploidy of the chromosome, and n is the number of long reads within a multiplicity block. The process of phasing each multiplicity block can be run in parallel, but in the current implementation, it is not parallelized. It took HAT less than an hour to run for each chromosome of *S.pastorianus*, because it has small multiplicity blocks. However, the *B.bruxellensis* took longer to run because it has longer multiplicity blocks (in some cases, chromosome scale). It took HAT 24 h to phase the longest chromosome of *B.bruxellensis*, which is 3.5 Mb. HAT memory usage is minimal; HAT uses <8 GB of memory for all datasets. We executed HAT on a system with 16 cores of CPU and 32 GB of memory. It is worth mentioning that HAT is a proof-of-concept implementation and is not optimized for speed.

4 Conclusion

HAT is a haplotype assembly tool that reconstructs haplotypes and phases genomes using NGS and TGS data. It is impossible to phase entire homologous chromosomes when there are large variation deserts. To address this, HAT identifies regions where some of the haplotypes are identical so they are taken into account when phasing. We show that NGS and TGS provide enough information to phase high-heterozygosity genomes on a chromosome scale and more than 90% of the alleles in low-heterozygosity genomes.

We evaluate the performance of HAT on six simulated datasets based on an aneuploid yeast strain *S.pastorianus* CBS1483, and compare it to nPhase and Whatshap, the state-of-the-art algorithms. We observe that HAT presents higher phasing accuracy, which results from starting with seeds created by accurate short reads. While all tools have decent performance in highly heterozygous genomes, HAT performs remarkably well in phasing and read clustering of low heterozygote genomes. However, in the latter case, haplotypes created by HAT are fragmented since it does not attempt to connect the multiplicity blocks, because there is not enough information to link them. While we did not evaluate HAT on any diploid dataset directly, we observed that HAT successfully phases blocks with a multiplicity level of 2 which shows that it can also be applied to diploid genomes.

The value of the distance_parameter affects HAT's result. A smaller distance_parameter affects HAT's result, for example, a smaller distance_parameter will lead to smaller multiplicity blocks. That means the final phasing is more accurate because only the seeds and the areas close to them are phased and connected. However, simultaneously, the final result is separated into more disjoint blocks because HAT considers multiplicity blocks disconnected from each other and never attempts to merge them. A larger distance_parameter will lead to larger multiplicity blocks, which means HAT connects areas that are further apart together. However, there will be less read support for some of these areas because they are far, meaning some areas remain unphased. On top of that, errors might affect the phasing if the distance_parameter is too large because there will be less read support, and the sequencing error might affect the majority voting. Based on our experience, the average read length of the long reads is a good trade-off between accuracy and the block length.

The main limitation of HAT is that it uses alignments of short and long reads to the reference. Similar to other haplotype assembly tools, HAT's performance greatly depends on the quality of this alignment and the subsequent variant calling. Moreover, HAT uses only the SNPs for phasing, thus it may not be able to reconstruct haplotypes in genomes with high levels of insertions, deletions and structural variations. Meanwhile, we do not expect different long reads error rates to affect HAT's accuracy, since HAT starts with seeds created using NGS accurate reads and requires high support at all steps.

Like all other reference-based haplotype reconstruction methods, HAT suffers from reference genome errors. Errors in the reference genome can lead to inaccurate variant calling, which immediately affects the haplotype reconstruction, as it is the primary source of information that HAT uses for the phasing. As an example, collapsed repeats can affect the ploidy estimation. The region with the collapsed repeat can have seeds with more combinations of alleles than the actual multiplicity of the region. That will create a small multiplicity block with a higher multiplicity than the region and lead to extra, wrong haplotypes for that region. However, it is worth mentioning that this region will be small because the seed with more combinations of alleles will not be joined with any other seed to create a more extended multiplicity block, and the multiplicity block will be around two times the distance_parameter.

Another potential limitation is that in rare cases, HAT may incorrectly assign a lower than actual multiplicity number. This occurs when there is a group of seeds in close proximity where the number of combinations of alleles in any of the seeds is smaller than the actual multiplicity level. When each seed is viewed separately, some of the haplotypes appear to be identical in that region. However, it is possible that these are different groups of identical haplotypes, and the ploidy level of the region may be higher if all of these seeds are viewed as a whole. This can be solved by creating seeds and identifying multiplicity blocks using long reads. However, considering all consecutive SNP loci in the reads as seeds requires significant computing power since each long read might cover hundreds of SNPs. Additionally, allelic combinations in the seeds may be affected by the high error rate of long reads. Another way to mitigate erroneous multiplicity assignment is to adjust multiplicity levels during the iterative part of HAT when long reads are used to phase the SNPs. In principle, by solving the mentioned problem it should be possible to create the seeds with HiFi reads, which will lead to longer multiplicity blocks and higher contiguity.

There are not many polyploid haplotype-resolved genomes at the chromosome scale, which hinders the development of novel haplotype assembly algorithms. Hence, haplotype simulators are limited and the simulated haplotypes differ significantly from the real ones. We observed this when we compared the multiplicity blocks of real and simulated data (compare Supplementary Figs S2 and S3). There are many regions in the real data where the multiplicity level is smaller than the chromosome's actual ploidy level, contrary to simulated data. This might change with HiC reads since they provide long-range information and link regions of the chromosome that are far apart. In addition to inconsistencies in the ploidy levels, the large variation deserts in CBS1483 genome cannot be simulated due to the limitations of current haplotype simulators.

Although we demonstrate the performance of HAT on only two yeast strains *S.pastorianus* CBS1483 and *B.bruxellensis* GB54, HAT can also phase different polyploid genomes. Since HAT performed

consistently well on various levels of ploidy and heterozygosity, we expect our results to generalize to other genomes of varying ploidy and that HAT can readily be adopted to different use cases. Moreover, we presume that HAT can find applications in metagenomics assembly since the haplotype and metagenomics assembly problems are comparable at the strain level. In metagenomics assembly, the goal is to reconstruct the genome of every single strain of the metagenomics community, which can be up to thousands of genomes. These strains, like haplotypes, are quite similar to each other. Furthermore, as a result of horizontal gene transfers, some of the species within the community share genomic content, complicating their read separation. Moreover, the sequencing coverage of strains varies significantly, which might lead to the underrepresented strains not being reconstructed in the assembly process.

Ultimately, HAT enables us to reconstruct haplotypes of polyploid genomes reliably, investigate the relationship of phenotypic features to the underlying haplotype alleles and gain a better understanding of genetic diversity.

Acknowledgements

We thank Erin Jordan and Stephanie Pillay for reviewing the article.

Financial support: none declared.

Conflict of Interest: none declared.

References

Abou Saada,O. *et al.* (2021) nPhase: an accurate and contiguous phasing method for polyploids. *Genome Biol.*, **22**, 126.

Bhat,J.A. *et al.* (2021) Features and applications of haplotypes in crop breeding. *Commun. Biol.*, **4**, 1–12.

Crawford,D.C. and Nickerson,D.A. (2005) Definition and clinical importance of haplotypes. *Annu. Rev. Med.*, **56**, 303–320.

Garg,S. (2021) Computational methods for chromosome-scale haplotype reconstruction. *Genome Biol.*, **22**, 1–24.

Huang,W. *et al.* (2012) ART: a next-generation sequencing read simulator. *Bioinformatics*, **28**, 593–594.

Li,H. (2018) Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, **34**, 3094–3100.

Moeinzadeh,M.-H. *et al.* (2020) Ranbow: a fast and accurate method for polyploid haplotype reconstruction. *PLoS Comput. Biol.*, **16**, e1007843.

Motazedi,E. *et al.* (2018) Exploiting next-generation sequencing to solve the haplotyping puzzle in polyploids: a simulation study. *Brief. Bioinform.*, **19**, 387–403.

Ng,S.B. *et al.* (2009) Targeted capture and massively parallel sequencing of 12 human exomes. *Nature*, **461**, 272–276.

Ramsey,J. and Schemske,D.W. (1998) Pathways, mechanisms, and rates of polyploid formation in flowering plants. *Annu. Rev. Ecol. Syst.*, **29**, 467–501.

Salazar,A.N. *et al.* (2019) Chromosome level assembly and comparative genome analysis confirm lager-brewing yeasts originated from a single hybridization. *BMC Genomics*, **20**, 1–18.

Schrinner,S.D. *et al.* (2020) Haplotype threading: accurate polyploid phasing from long reads. *Genome Biol.*, **21**, 1–22.

Vaser,R. *et al.* (2017) Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Res.*, **27**, 737–746.

Walker,B.J. *et al.* (2014) Pilon: an integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PLoS One*, **9**, e112963.

Wick,R. (2019) Badread: simulation of error-prone long reads. *J. Open Source Softw.*, **4**, 1316.