HaploTypo v1.0

User manual

Authors and maintainers: Cinta Pegueroles, Laia Carreté, Verónica Mixão, Manu Molina, and Toni Gabaldón

Contact: cinta.pegueroles@bsc.es / laia.carrete@gmail.com / vmixao@gmail.com / manu.molinam@gmail.com / toni.gabaldon.bcn@gmail.com

November 23rd, 2019

1. Description

With recent advances in Next Generation Sequencing and bioinformatics, the number of available phased genome assemblies (i.e. those in which diploid genome sequences are resolved at the haplotype level) is increasing. However, as current tools for variant calling do not take into account the presence of two haplotypes in the same reference genome, the user has to choose between the following approaches: i) assess the heterozygosity of the genome, and detect heterozygous positions without phasing information (i.e. by using only one haplotype as reference), or ii) obtain information regarding the variability in each haplotype but do not get the heterozygosity levels (i.e. use a phased genome as reference). HaploTypo performs read mapping and variant calling on diploid phased genomes, providing information not only of the variability levels of the analyzed sample, but also of the alternative variants for each haplotype. The pipeline is divided into four modules: (1) read mapping to each of the haplotypes using BWA-MEM (Li, 2013), (2) variant calling using GATK (McKenna *et al.*, 2010), bcftools (Li, 2011) or freebayes (Garrison and Marth, 2012), (3) inference of the true alternative variants for each haplotype, and (4) reconstruction of the final alternative haplotypes. Each of these parts constitute a separated python script. Hence, if the user is interested in only one of the modules, this can be run independently.

2. Installation

2.1. From a GitHub repository:

1. Download the pipeline from GitHub
   git clone https://github.com/Gabaldonlab/haplotypo.git

2. Make sure that you have the following dependencies installed in your machine
   a. Java version 1.8
   b. Python 2.7 or 3.5
   c. pyvcf package v. 0.6.7 (https://pyvcf.readthedocs.io/en/latest/) -> necessary to run module 3
   d. BWA-MEM v0.7.15 (Li 2013) -> necessary to run module 1
   e. Samtools v1.9 (Li et al. 2009) -> necessary to run module 2
   f. Picard v4.0.2.1 (http://broadinstitute.github.io/picard) -> necessary to run modules 1, 2
   g. Bcftools v1.9* (Li, 2011) -> necessary to run module 2
   h. GATK v4.0.2.1* (McKenna et al. 2010) -> necessary to run module 2

i. Freebayes v1.1* (Gabor and Marth 2012) -> necessary to run module 2
j. Vcflib v1.0.0-rc0-333-g5b0f (https://github.com/vcflib/vcflib) -> necessary to run freebayes in module 2
** You can choose one of these programs to do variant calling*

3. Use one of the install scripts in the "scripts" folder. There are two versions, for python 2.7 and 3.5. Both scripts install dependencies and a HaploTypo alias for the selected Python version.

4. Change the path of the programs in the `programs_config.py` file if necessary (optional step).

2.2. Run HaploTypo in a Docker container:

1. Pull the Docker image with HaploTypo to use the last version
   `docker pull cgenomics/haplotypo`

2. To test the Haplotypo Docker container
   ```
   docker run -it cgenomics/haplotypo python
   /root/src/haplotypo/bin/haplotypo.py
   ```

3. To test the image with test data, Process the data inside the image - all data will be lost at the end
   ```
   docker run -it cgenomics/haplotypo python
   /root/src/haplotypo/bin/haplotypo.py -o ./shared/output -amb
   0 -hapA ./dataset/A.chr7.fa -hapB ./dataset/B.chr7.fa -idA
   A_AB -idB B_AB -f1 ./dataset/AB.chr7.1.fq.gz -f2
   ./dataset/AB.chr7.2.fq.gz -caller GATK
   ```

4. To test the image with real data. Replace {mydataset} and parameters with your dataset folder structure. It will create an output folder with the result files.
   ```
   docker run -v
   `pwd`/{mydataset}/:/root/src/haplotypo/dataset:rw \
       -it cgenomics/haplotypo \
       python /root/src/haplotypo/bin/haplotypo.py \
       -o ./dataset/output \
       -amb 0 \
       -hapA ./dataset/{myfileA.fa} \
       -hapB ./dataset/{myfileB.fa} \
       -idA A_AB \
       -idB B_AB \
       -f1 ./dataset/{myfileAB.1.fq.gz} \
       -f2 ./dataset/{myfileAB.2.fq.gz} \
       -caller GATK
   ```

5. If you want to know more about this process.

3. Usage

HaploTypo can be used in two ways: i) running the entire pipeline using the script `haplotypo.py`; ii) or running independently each of the different 4 modules. These modules are (Figure 1):

1. Read mapping (`mapping.py`)
2. Variant calling (`var_calling.py`)
3. Inference of true alternative variants for each haplotype (`VCFcorr_alleles.py`)
4. Reconstruction of phased haplotypes (`haplomaker.py`)

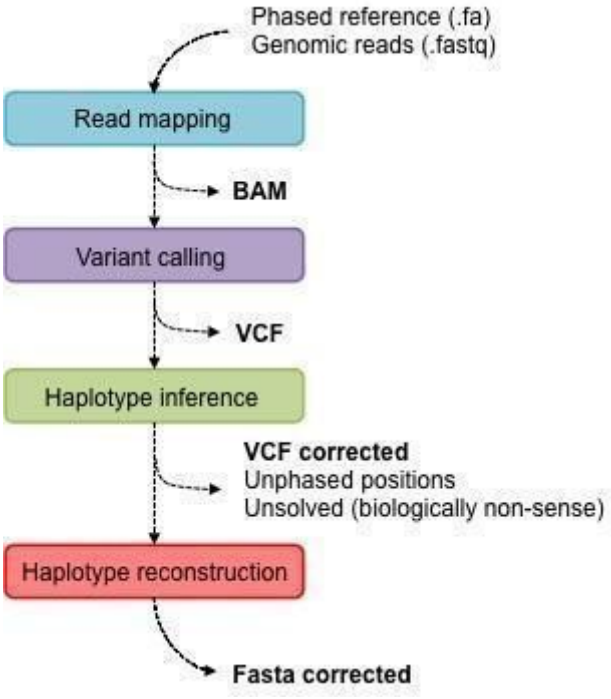

Figure 1. Schematic representation of HaploTypo pipeline.

Please note that `mapping.py` and `var_calling.py` scripts can be modified by users at their own risk to adjust the parameters of read mapping and variant calling, respectively.

3.1. Read mapping (`mapping.py`)

The first part of HaploTypo is a read mapping step using `mapping.py` (Figure 1). This script takes as input the two haplotypes of a phased reference in two separate FASTA files and the filtered genomic paired-end reads in FASTQ format. Alternatively, and only in the case the user does not want to run the whole pipeline, a single haploid reference can be provided. Then, BWA-MEM is used to align the sequencing reads to each reference fasta file. Picard is afterwards required to sort, mark duplicates and index the obtained BAM file(s),

and to finally check some quality metrics of the alignment. The main output of this module is one or two BAM files (depending if a haploid or diploid reference was provided) with the read alignment, which will then be used as input in module 2. Detailed information on the output of this module can be obtained in the Output section.

Arguments:

-o, --outdir Directory where the data will be stored
-thr, --threads Number of threads [default: 8]
-hapA, --hapA Haplotype A from reference genome (fasta)
-hapB, --hapB Haplotype B from reference genome (fasta)
-ref, --reference Unphased reference genome (fasta). Only required if no phased haplotypes are provided
-idA, --TagName_hapA Tag to be added in the output files relative to hapA. [default: hapA]
-idB, --TagName_hapB Tag to be added in the output files relative to hapB. [default: hapB]
-t, --TagName Tag to be added in the output files if using a non-phased reference. Only required if no phased haplotypes are provided. [default: strain]
-f1, --fastq1 Illumina paired-end reads 1
-f2, --fastq2 Illumina paired-end reads 2

Examples:
Read mapping on phased haplotypes

```
python mapping.py -o out_dir -thr 8 -hapA hapA.fa -hapB hapB.fa -idA hapA -idB hapB -f1 reads.1.fq -f2 reads.2.fq
```

Read mapping on unphased genome

```
python mapping.py -o out_dir -thr 8 -ref ref.fa -t strain -f1 reads.1.fq -f2 reads.2.fq
```

3.2. Variant calling (`var_calling.py`)

The second part of HaploTypo is a variant calling step using `var_calling.py` script (Figure 1). This script takes as input one or two BAM files with the sequencing reads aligned to the haploid reference genome or each of the phased haplotypes, depending if a haploid or diploid reference is used (check the arguments list), and the respective FASTA file(s). Variant calling and filtration are performed with GATK , Bcftools or Freebayes, according to the user's choice. Each program is executed using the parameters recommended by their respective authors. The main output of this module is one or two VCF files (depending if a haploid or diploid reference was provided) with all SNPs passing the filtration process. In case a diploid reference was used, the two VCFs can be used as input for module 3. Detailed information on the output of this module can be obtained in the Output section.

Please be aware that the option to change the ploidy level should only be used if you **DO NOT** want to use `VCFcorr_alleles.py` or `haplomaker.py`!

Arguments:

-o, --outdir Directory where the data will be stored
-thr, --threads Number of threads [default: 8]
-hapA, --hapA Haplotype A from reference genome (fasta)
-hapB, --hapB Haplotype B from reference genome (fasta)
-ref, --reference Unphased reference genome (fasta). Only required if no phased haplotypes are provided
-idA, --TagName_hapA Tag to be added in the output files relative to hapA. [default: hapA]
-idB, --TagName_hapB Tag to be added in the output files relative to hapB. [default: hapB]
-t, --TagName Tag to be added in the output files if using a non-phased reference. Only required if no phased haplotypes are provided. [default: strain]
-bA, --bamA Bam file using haplotype A as reference
-bB, --bamB Bam file using haplotype B as reference
-b, --bam Bam file using a non-phased reference. Only required if no phased haplotypes are provided
-c, --coverage Minimum coverage necessary for variant calling [default: 30]
-caller, --variant_caller Program for variant calling (GATK/bcftools/freebayes). [default: GATK]
-p, --ploidy Ploidy [default: 2]
-pfile, --ploidy_file Ploidy file. Only necessary for variant calling with bcftools and ploidy != 2

Examples:

Variant calling using phased haplotypes

```
python var_calling.py -o out_dir -thr 8 -hapA hapA.fa -hapB hapB.fa -idA hapA -idB hapB -bA hapA.mkd.bam -bB hapB.mkd.bam -c 30 -caller GATK -p 2
```

Variant calling using unphased genome

```
python var_calling.py -o out_dir -thr 8 -ref ref.fa -t strain -b strain.mkd.bam -c 30 -caller GATK -p 2
```

3.3. Infer alternative variants for each haplotype (`VCFcorr_alleles.py`)

The third module in HaploTypo (`VCFcorr_alleles.py`, Figure 1) compares the variant

calling results for a given sample when it was mapped independently against each of the phased haplotypes. Thus, this module takes as input the two reference haplotypes, which should be provided as independent FASTA files, and the respective VCF files, containing only SNPs passing the filtration requirements. The default behavior of this module assumes that the two haplotypes have a 1-to-1 position correspondence. If this is not the case, a conversion table should also be provided (check arguments list). This table is only required if the two copies of each chromosome do not have a 1-to-1 position correspondence due to the presence of INDELs. It is a tab separated file in which, in each line, there is the name of the chromosome of haplotype A and B and the nucleotide position of chromosome A and B respectively. In the end this module generates two VCF files, one for each haplotype, reporting the variants specifically observed in each of them (Table 1, Additional table 1). In situations where it is not clear which haplotype presents a given variant, one of three options can be chosen to output this position: i) ignore the variant and do not report it; ii) report this position in both VCF files with IUPAC ambiguity code; iii) randomly assign the variant to one of the haplotypes. Detailed information on the output of this module can be obtained in the Output section.

Please be aware that if you are running each module independently and not the `haplotypo.py`, to run this script you need to have run `var_calling.py` script considering phased haplotypes and ploidy 2, as it only considers diploid samples.

Table 1. Explanation on how VCFcorr_alleles.py compares and reports the final VCF. For each variable position, the allele present in each reference haplotype is compared with the individual genotype. From this comparison, we infer which allele is present in each of the haplotypes of the analyzed individual. 0 - allele similar to the reference; 1 - allele different from the reference; 2 - allele different from the reference and from allele 1. It is important to note that some of the cases described in this table are rare. For instance, most of the 1/1 - 1/1 cases will be solved, with few exceptions. See Additional Table 1 for detailed examples for each case.

Genotype haplotype A	Genotype haplotype B	corrected VCF A	corrected VCF B
1/1	1/1	1 / unsolved	1 / unsolved
0/1	1/2	0 / unsolved	1 / unsolved
1/2	0/1	1 / unsolved	0 / unsolved
0/1	0/1	0 / unphased / unsolved	0 / unphased / unsolved
1/2	1/2	unphased / unsolved	unphased / unsolved
1/1	No SNP	phased / unsolved	phased / unsolved
0/1	1/1	unsolved	unsolved
1/1	0/1	unsolved	unsolved
1/1	1/2	unsolved	unsolved
1/2	1/1	unsolved	unsolved
0/1	No SNP	unsolved	unsolved
1/2	No SNP	unsolved	unsolved

Arguments:

-A, --VCF_A VCF with PASS SNPs obtained after running a variant caller using HapA as reference
-B, --VCF_B VCF with PASS SNPs obtained after running a variant caller using HapB as reference
-fastaA, --fastaA Fasta with HapA used as reference for variant calling
-fastaB, --fastaB Fasta with HapB used as reference for variant calling
-cA, --VCF_corrA Out file for the corrected VCF A
-cB, --VCF_corrB Out file for the corrected VCF B
-amb, --ambiguity Define how to proceed with ambiguous genotypes. We offer 3 options: 0, no print; 1, print ambiguity codes; 2, randomly assign a nucleotide [default: 0]
-coor, --coordinatesTable Coordinates table. Required only if the two copies of each chromosome have different lengths. Format: tab separated table with chr_hapA"\t"chr_HapB"\t"positionA"\t"positionB; ex: chrRA chrRB 1 1 [optional]

WARNING! Chromosome Id must be /(chr.*?)_.*/; ex: chr1_A & chr1_B; chr11_A & chr11_B; chrR_A & chrR_B;

Please be aware that -amb1 option outputs in vcfv4.3 format, which is the only one that allows ambiguity info. This format is not readable with igv but yes with bctools; ex: bcftools query -f '%CHROM %POS %REF %ALT{0}\n' file_amb1.vcf

Example:

```
python VCFcorr_alleles.py -A hapA.pass.snp.vcf -B hapB.pass.snp.vcf -fastaA hapA.fasta -fastaB hapB.fasta -cA hapA.corrected_amb0.vcf -cB hapB.corrected_amb0.vcf -amb 0
```

3.4. Reconstruction of phased haplotypes (`haplomaker.py`)

The fourth module of HaploTypo (`haplomaker.py`, Figure 1) takes as input the two VCF files obtained by `VCFcorr_alleles.py` script (one for each haplotype), and the respective reference haplotypes in FASTA format. Then, it changes each position in the respective reference haplotype by its corrected nucleotide, generating a FASTA file for each reconstructed haplotype.

Arguments:

-o, --outdir Directory where the data will be stored
-hapA, --hapA Haplotype A from reference genome (fasta)
-hapB, --hapB Haplotype B from reference genome (fasta)
-corrA, --corrected_A VCF file with the corrected variants for haplotype A
-corrB, --corrected_B VCF file with the corrected variants for haplotype B

Example:

```
python haplomaker.py -o out_dir -hapA hapA.fa -hapB hapB.fa -corrA hapA.corrected_amb0.vcf -corrB hapB.corrected_amb0.vcf
```

3.5. Using the full HaploTypo pipeline (`haplotypo.py`)

As mentioned before the whole pipeline can be run at once, using the `haplotypo.py` script. In this case, it is mandatory to provide two FASTA files as reference, one for each haplotype. Furthermore, paired-end libraries in FASTQ format, or alternatively, their alignment to each of the haplotypes in BAM format are required (check arguments list). In this last case, the HaploTypo does not run module 1, and therefore the outputs of this module are not generated.

Arguments:

`-o, --outdir` Directory where the data will be stored
`-thr, --threads` Number of threads [default: 8]
`-c, --coverage` Minimum coverage necessary for variant calling [default: 30]
`-amb, --ambiguity` Define how to proceed with ambiguous genotypes. We offer 3 options: 0, no print; 1, print ambiguity codes; 2, randomly assign a nucleotide [default: 0]
`-hapA, --hapA` Haplotype A from reference genome (fasta). Warning: the name of each chromosome should be chr* in both haplotypes
`-hapB, --hapB` Haplotype B from reference genome (fasta). Warning: the name of each chromosome should be chr* in both haplotypes
`-idA, --TagName_hapA` Tag to be added in the output files relative to hapA [default: hapA]
`-idB, --TagName_hapB` Tag to be added in the output files relative to hapB [default: hapB]
`-caller, --variant_caller` Program for variant calling (GATK / bcftools / freebayes) [default: GATK]
`-f1, --fastq1` Illumina paired-end reads 1 (only required for read mapping) [optional]
`-f2, --fastq2` Illumina paired-end reads 2 (only required for read mapping) [optional]
`-bA, --bamA` Bam file using haplotype A as reference [optional]
`-bB, --bamB` Bam file using haplotype B as reference [optional]
`-coor, --coordinatesTable` Coordinates table. Required only if the two copies of each chromosome have different lengths. Format: tab separated table with chr_hapA"\t"chr_HapB"\t"positionA"\t"positionB; ex: chrRA chrRB 1 1 [optional]

Note: Although they are tagged as optional parameters, haplotypo.py always requires the sequencing reads or their respective alignment to each of the haplotypes!

4. Output

The output files generated by the different modules of HaploTypo are described in Table 2.

Table 2. Output files obtained with HaploTypo, with indication of file name, file description and respective module generating the file.

Module	File name	Description
Module 1	sample.mkd.bam	Bam file with read alignment against the respective haplotype.
Module 1	sample.mkd.bai	Bam index file.
Module 1	sample.mkd.metric.txt	Txt file with the metrict generated form MarkDuplicates function from Picard.
Module 1	sample.mkd.stat.txt	Txt file with general mapping stats generated from CollectAlignmentSummaryMetrics function from Picard.
Module 2	sample.GATK.bam	Bam file generated by HaplotypeCaller tool from GATK (only generated if -caller is set to GATK).
Module 2	sample.GATK.bai	GATK bam index file (only generated if -caller is set to GATK).
Module 2	sample.vcf	VCF file with all the variants called against the respective haplotype (not generated if -caller set to freebayes).
Module 2	sample.vcf.idx	Index file for the raw VCF file.
Module 2	sample.flt.vcf	Raw VCF file with an extra column saying if the variants passed the filters indicated for the filtration process (valid for GATK and bcftools). If -caller is set to freebayes this is the first file generated after the calling, which already considers some filters. In this case this extra column is presented, but all variants present a dot ".". Labels applied by GATK: PASS, variants passed all the filters; BadDepthofQualityFilter, variant does not pass the required quality options.
Module 2	sample.flt.vcf.idx	Index file for the VCF file with filtering labels.
Module 2	sample.flt.snp.vcf	VCF file containing only SNPs with filtering labels (only generated by GATK).
Module 2	sample.flt.snp.vcf.idx	Index file for the VCF file with filtering labels containing only SNPs.
Module 2	sample.pass.snp.vcf	VCF file only with SNPs that passed all the filters (these are the variants considered for further analysis).
Module 2	sample.mpileup.bcf	Compressed BCF file with information for each position in the reference (only generated by bcftools).
Module 3	sample.corrected_amb0.vcf	VCF file with the corrected variants for the

		respective haplotype. All the variants here reported belong to the respective haplotype and not to the alternative (ambiguous positions were excluded).
Module 3	sample.corrected_amb1.vcf	VCF file with the corrected variants for the respective haplotype. All the variants here reported belong to the respective haplotype and not to the alternative (ambiguous positions were solved using ambiguity codes).
Module 3	sample.corrected_amb2.vcf	VCF file with the corrected variants for the respective haplotype All the variants here reported belong to the respective haplotype and not to the alternative (ambiguous positions were solved randomly attributing each of the alternatives to each haplotype).
Module 3	sample.unphasedVariants_ambX.bed	Bed file with positions that were not possible to phase. Please note that these positions are included in the sample.corrected_ambX.vcf files. Ambiguity can be set to 0 (do not print ambiguity positions), 1 (ambiguity codes) or 2 (random assignation).
Module 3	sample.unsolved_ambX.bed	Bed file with positions that were not possible to solve since they are most likely the result of alignment or calling errors. Ambiguity can be set to 0 (do not print ambiguity positions), 1 (ambiguity codes) or 2 (random assignation).
Module 4	sample.alternative.fasta	Fasta file with the reconstructed haplotype.

5. Examples

5.1. Run the complete pipeline using fastq files

```
python haplotypo.py -o path/to/output/ -thr 8 -c 30 -amb 0 -hapA
reference_genome_hapA.fasta -hapB reference_genome_hapB.fasta -idA
name_example_hapA -idB name_example_hapB -f1 reads_1.fastq.gz -f2
reads_2.fastq.gz -caller GATK
```

5.2. Run the pipeline when you have your bam files (`haplotypo.py` will not run `mapping.py`)

```
python haplotypo.py -o path/to/output/ -thr 8 -c 20 -amb 1 -hapA
```

```
reference_genome_hapA.fasta -hapB reference_genome_hapB.fasta -idA
name_example_hapA -idB name_example_hapB.bam -bA sample_hapA.bam
-bB sample_hapB.bam -caller freebayes
```

5.3. Run the pipeline when the two haplotypes do not have a 1-to-1 position correspondence

```
python haplotypo.py -o path/to/output/ -thr 8 -c 20 -amb 0 -hapA
reference_genome_hapA.fasta -hapB reference_genome_hapB.fasta -idA
name_example_hapA -idB name_example_hapB -f1 reads_1.fastq -f2
reads_2.fastq -caller bcftools -coor coordinates_table.txt
```

6. Description of the algorithm

HaploTypo starts the phasing step by running the `VCFcorr_alleles.py` script. This module requires as input the two reference haplotypes, which should be provided as independent FASTA files, and the respective VCF files, containing only SNPs passing the filtering requirements. The default behavior of this module assumes that the sequences of the two haplotypes have a 1-to-1 position correspondence (i.e. the residue at position *i* in the first haplotype is the allele of the position *i* in the second haplotype). If this is not the case, a conversion table between the coordinates of the two haplotypes should also be provided (check arguments list). This table is only required if the two copies of each chromosome do not have a 1-to-1 position correspondence due to the presence of INDELs. The correspondence table is a tab separated file in which, in each line, there is the name of the chromosome of haplotype A and B and the nucleotide position of chromosome A and B respectively.

The script first checks for the presence of the coordinates table. If no coordinates table was provided, it reads the two VCF files provided for haplotypes A and B, and stores their information in two dictionaries. If the user provided a table, it stores the conversion table in memory and it reads the VCF file for haplotype A and stores its information and the respective position correspondence in haplotype B in a dictionary. Please note that the program will print a warning if a variable position is missing in the coordinates table. Then, it reads the VCF file for haplotype B and stores its information in a dictionary.

Once variants have been stored in two dictionaries (for haplotypes A and B), `VCFcorr_alleles.py` script calls the VCFcorr function to phase haplotype A and B independently and to obtain the corrected VCF for each of them. This function aims to phase every variant in a given haplotype by comparing with the variant called in the corresponding position in the other haplotype. In the case that no SNP was called in the other haplotype, it retrieves the nucleotide from the reference sequence of that haplotype. The function assess the two identities (the two alleles) of that site and the two haplotype reference sequences,

and decides the most likely phasing output. All possible comparisons handled by the VCFcorr function are detailed in Table 1, which also provides details of the decisions taken in each case. For instance, in case 1 we consider a 0/1 variant for haplotype A, where A is the reference and G the alternative allele. In the corresponding position for haplotype B it was called a 0/1 variant, being G the reference and A the alternative allele. Thus, we can infer that our sample has A in haplotype A and G in haplotype B, and therefore the position is not reported in the corrected VCF files. In case 2 we can not assign alleles to each haplotype (unphased) since the nucleotide in the two reference haplotypes is the same (G). In the unphased cases, the script will check which ambiguity option was selected. If ambiguity was set to 0 the unphased variant is only reported in a BED file containing all the unphased positions. If set to 1, it will print an IUPAC ambiguity code in the corrected VCF. If set to 2, it will print one of the two possible nucleotides randomly in the corrected VCF (in case 2, it will print randomly A or G). Phasing results are stored in two independent dictionaries for haplotype A and B and printed in the VCF corrected file. It is important noting that in the corrected VCF only solved cases with a corrected allele different from the reference (GT=1) are printed and, in general, most of the solved cases have the same nucleotide as the reference (GT=0). In case 3, the variant remains unsolved since the genotypes called for each haplotype are not compatible (GA for haplotype A and GT for haplotype B). Thus, this variant will be only printed in the BED file containing all the unsolved positions. At the end, three different types of output are expected from this module: i) two VCF files with the corrected variants for each of the haplotypes; ii) a BED file with all the unphased positions; iii) a file containing the unsolved positions (i.e. positions where the genotypes called in each reference haplotype are incompatible). The two VCF files with the corrected alleles for each haplotype can be given as input for `haplomaker.py`. Detailed information on its usage and outputs generated by this module can be obtained in sections 3 and 4.

7. Limitations

Please be aware that HaploTypo assumes that organisms are diploid. Different ploidy levels can only be used for unphased reference genomes, and only for the first two modules.

8. Liability and disclaimer

HaploTypo is free and comes without any warranty. Users should use it at their own risk. We cannot assure that it will fulfill specific needs.

9. Citation

C. Pegueroles, V. Mixão, L. Carreté, M. Molina, and T. Gabaldón. HaploTypo: a variant-calling pipeline for phased genomes. Under revision.

HaploTypo uses previously published tools for mapping and variant calling. Therefore, we encourage users to cite the programs they used in these steps in addition to HaploTypo citation (check the References section).

10. Additional material

Test data set: provided in the data folder included in the HaploTypo docker container (see installation above) and in the GitHub repository (https://github.com/gabaldonlab/haplotypo).

Additional table 1. Table listing all possible cases that may be encountered by HaploTypo when evaluating variants called in each of the two haplotypes, and how they are handled and encoded in the output. After read mapping and variant calling to the two phased reference haplotypes separately (modules 1 and 2), HaploTypo outputs the results in two files with the '.pass.snp.vcf' extension. Then, HaploTypo infers the haplotype correspondence for each variant (module 3). **White** and **grey** background colors in the table represent the two haplotypes in each case scenario. **Ref:** reference allele provided by 'sample.pass.snp.vcf' files, or by 'haplotype.fasta' file in the absence of called SNP in one of the haplotypes; **Alt:** alternative allele provided by 'sample.pass.snp.vcf' files; **GT:** genotype provided by 'sample.pass.snp.vcf' files; **iAlt:** alternative allele inferred by HaploTypo module 3 for each haplotype of an individual different from the reference; **Category**: "phased" - HaploTypo can assign an allele to each haplotype, "unphased" - HaploTypo cannot assign an allele to each haplotype, or "unsolved" - HaploTypo found an incompatibility in the alleles or genotypes reported for each haplotype; **Reported:** "no" - position and alternative allele for this haplotype are not reported, 'sample.corrected_amb1/2.vcf' - file where alternative alleles are reported using an ambiguity code or random assignation (-amb 1 and -amb 2 options respectively, depending on the user's choice), 'sample.unphasedVariants_amb0/1/2.bed' - file where unphased positions are reported, 'sample.unsolved_amb0/1/2.bed' - file where unsolved positions are reported.

Cases	Input			Output		
	Ref	Alt	GT	iAlt	Category	Reported
case 1	A	G	0/1	A	phased	no
	G	A	0/1	G	phased	no
case 2	G	A	0/1	R	unphased	sample.corrected_amb1/2.vcf sample.unphasedVariants_amb0/1/2.bed

	G	A	0/1	R	unphased	sample.corrected_amb1/2.vcf sample.unphasedVariants_amb0/1/2.bed
case 3	A	G	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	G	T	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
case 4	A	C	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	G	T	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
case 5	A	T	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	A	G	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
case 6	G	C, A	1/2	M	unphased	sample.corrected_amb1/2.vcf sample.unphasedVariants_amb0/1/2.bed
	G	C, A	1/2	M	unphased	sample.corrected_amb1/2.vcf sample.unphasedVariants_amb0/1/2.bed
case 7	G	C, A	1/2	M	unphased	sample.corrected_amb1/2.vcf sample.unphasedVariants_amb0/1/2.bed
	T	C, A	1/2	M	unphased	sample.corrected_amb1/2.vcf sample.unphasedVariants_amb0/1/2.bed
case 8	A	C,T	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
	A	C,G	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
case 9	A	C,T	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
	T	C,G	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
case 10	A	C,T	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
	T	A,C	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
case 11	T	C	1/1	C	phased	sample.corrected_amb1/2.vcf
	T	C	1/1	C	phased	sample.corrected_amb1/2.vcf
case 12	A	C	1/1	C	phased	sample.corrected_amb1/2.vcf
	T	C	1/1	C	phased	sample.corrected_amb1/2.vcf
case 13	T	G	1/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	T	C	1/1	-	unsolved	sample.unsolved_amb0/1/2.bed
case 14	A	T	1/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	T	A	1/1	-	unsolved	sample.unsolved_amb0/1/2.bed
case 15	A	C	1/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	T	A	1/1	-	unsolved	sample.unsolved_amb0/1/2.bed
case 16	A	G	0/1	A	phased	no
	T	A,G	1/2	G	phased	sample.corrected_amb1/2.vcf
case 17	T	A	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed

	A	T,C	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
case 18	A	T	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	T	C,G	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
case 19	A	T	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	A	T,G	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
case 20	A	T	0/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	A	C,G	1/2	-	unsolved	sample.unsolved_amb0/1/2.bed
case 21	A	T	1/1	T	phased	sample.corrected_amb0/1/2.vcf
	T	-	-	T	phased	no
case 22	A	C	1/1	-	unsolved	sample.unsolved_amb0/1/2.bed
	T	-	-	-	unsolved	sample.unsolved_amb0/1/2.bed

References

Gabor,G. and Marth,E. (2012) Haplotype-based variant detection from short-read sequencing. *arXiv*:1207.3907.

Li,H. (2013) Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*:1303.3997v2

Li,H. (2011) A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, **27**, 2987–2993.

Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

McKenna,A. *et al.* (2010) The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**, 1297–1303.