Fast and accurate imputation of genotypes from noisy low-coverage sequencing data in bi-parental populations

Cécile Triay[1,§], Alice Boizet[2,§], Anestis Gkanogiannis[3], Christopher Fragoso[4,5], Jean-François Rami[2], Mathias Lorieux[1,3,*]

[1]University of Montpellier, DIADE, IRD, France

[2]AGAP, Univ Montpellier, CIRAD, INRA, Montpellier SupAgro, Montpellier, France

[3]Agrobiotechnology Unit, Alliance Bioversity-CIAT, International Center for Tropical Agriculture, Cali, Colombia.

[4]Verinomics, Inc., 5 Science Park, New Haven, CT 06511, USA

[5]Department of Molecular, Cellular, and Developmental Biology, Yale University, New Haven, Connecticut 06511

[*]Corresponding author

[§]Equivalent contribution

Abstract

Motivation: Bi-parental populations genotyping can be performed with low-coverage next-generation sequencing (LC-NGS). This allows the creation of highly saturated genetic maps at reasonable cost, precisely localized recombination breakpoints, and minimized mapping intervals for quantitative-trait locus analysis.

The main issues with these low-coverage genotyping methods are (1) poor performance at heterozygous loci, (2) a high percentage of missing data, (3) local errors due to erroneous mapping of sequencing reads and reference genome mistakes, and (4) global, technical errors inherent to NGS itself.

Recent methods like Tassel-FSFHap or LB-Impute are excellent at addressing issues 1 and 2, but nonetheless perform poorly when issues 3 and 4 are persistent in a dataset (i.e., "noisy" data). Here, we present an algorithm for imputation of LC-NGS data that eliminates the need of complex pre-filtering of noisy data, accurately types heterozygous chromosomic regions, corrects erroneous data, and imputes missing data. The imputation of genotypes and recombination breakpoints is based on maximum-likelihood estimation. We compare its performance with Tassel-FSFHap and LB-Impute using simulated data and three real datasets. Furthermore, the algorithm is much faster than the Hidden Markov-Chains method.

Availability: NOISYmputer is available as a multiplatform (Linux, macOS, Windows) Java executable at the URL https://gitlab.cirad.fr/noisymputer/noisymputerstandalone/-/tree/1.0.0-RELEASE?ref_type=tags. The source code is available at the same URL.

Introduction

In genetic studies, bi-parental genetic populations can be created from inbred parental lines using various crossing systems, e.g., F_2 intercross issued from F_1 self-pollination (F_2) and recombinant inbred lines by single seed descent (SSD). These populations are used to create recombination maps and, if phenotypes are available, to find gene or quantitative-trait locus (QTL) genomic positions.

To do so, each individual of the population under study has to be characterized for its genomic content – or "genotyped" at many loci. This can be done using different molecular biology techniques, including various types of molecular markers. The gold standard for genetic variant discovery is obtained by different next-generation sequencing (NGS) techniques like restriction site-associated DNA sequencing (RADseq) (Davey and Blaxter 2010), genotyping by sequencing (GBS) (Elshire *et al.* 2011), and whole-genome sequencing (WGS) (Huang *et al.* 2009). These techniques provide very large numbers of markers and therefore facilitate the construction of highly saturated genetic maps. This provides accurate locations of

recombination breakpoints in each individual, which is important for a number of applications, e.g., studies of local recombination rate, genetic maps comparison, or QTL detection. Though NGS is less and less expensive to implement, sequencing a large number of samples can still be costly, and is commonly applied via reduced representation (RRS-NGS) or low-coverage (LC-NGS) strategies to reduce genotyping costs.

Reducing sequencing costs through minimized per-sample coverage has an important experimental downside: LC-NGS mechanically introduces a series of issues, the main ones being:

- **Issue 1:** *Low power to detect heterozygosity under low coverage*: For example, if only one sequencing read is generated at a locus, only one of the two alleles is revealed. As each additional read has a 0.5 probability of detecting the second allele, even 3 reads have only 0.25 probability of failing to detect a heterozygous call. Spread over thousands of sites, extensive inaccuracy in heterozygous regions becomes highly problematic.
- **Issue 2:** *Extensive genotype missingness*: The sparse distribution of reads at low coverage (3X coverage, for example, only implies an *average* of 3 reads per site) results in a complete lack of reads at some variant loci. Even in plants, which contain more genetic variation than humans, there are 6-22 SNPs per 1 Kb, resulting in abundant opportunity for non-reference variant missingness under low coverage (Xu *et al.* 2017).
- **Issue 3:** *Errors due to erroneous mapping of sequencing reads*: NGS technologies are based on short reads (e.g., 150 base pair, paired-end Illumina technology). Due to the combinatorial limitation of the sequence contained in short reads, multiple mapping locations may be identified, especially in plant genomes which exhibit much more repetitive content than human genomes. Additionally, in plants, such as rice, it has been shown that structural variation specific to subpopulations may be completely missing in any single reference genome. These assembly errors, omissions, and challenges posed by repetitious regions are sources of erroneous variants. Moreover, outright assembly errors may cause consistent, yet locally encountered genotyping errors.
- **Issue 4:** *Technical errors inherent to NGS methodology*: Sequencing errors may be globally introduced at a variety of stages in the NGS pipeline, from errors incurred in PCR-dependent library construction to NGS sequencing itself. The initial GBS protocol is known to generate libraries contaminated by chimeric inserts (Heffelfinger *et al.* 2014). Although rare, these errors may become problematic at low coverage, as additional reads refuting an erroneous call may not be available at a given locus.

Common imputation algorithms implemented in computer programs like Beagle (Browning and Browning 2007; Browning *et al.* 2021) or Impute2 (Howie *et al.* 2012), although very accurate in diversity panels, are not well adapted to the bi-parental context since they rely on large databases to infer haplotypes. Efficient methods have been recently developed to impute genotypic data derived from LC-NGS assays in bi-parental populations. For instance, Tassel-FSFHap (Swarts *et al.* 2014) and LB-Impute (Fragoso *et al.* 2016) can all address issues 1 and 2 accurately. Yet, these methods can produce inaccurate results when the errors mentioned in issues 3 and 4 – thereafter called "noisy data" – are too frequent. Thus, these methods might require additional bioinformatic steps to filter out low-quality markers before and after imputation. Even then, troublesome markers might not be detected easily and could alter dramatically the quality of the imputation and the final genetic map.

In this work, we present NOISYmputer, a maximum likelihood estimation algorithm for imputation of LC-NGS data that eliminates the need of complex pre-filtering of noisy data, accurately finds heterozygous chromosomic regions, corrects erroneous data, imputes missing data and precisely locates the recombination breakpoints (or crossovers). We test its accuracy using simulated data and we compare its performance with Tassel-FSFHap, LB-Impute using three datasets: (1) a rice F2 population sequenced in WGS, (2) a maize F2 population sequenced in GBS and (3) 84 simulated VCFs of F2 population with controlled depth, error rate and marker density. The algorithm is implemented in NOISYmputer, a multiplatform Java command line program (see "Availability" section).

Design and implementation

Imputation method

In this section we describe the main imputation algorithm, which is applied separately to each chromosome. The imputation can be preceded or followed by different filtering options in NOISYmputer (details in next section) that can be applied to reduce or eliminate the noise in the data (Figure 1).

By imputation we mean here guessing, confirming or correcting the genotype at a SNP site in a sample. LC-NGS generates poor information in heterozygous regions (see explanation on the confounding effect in SNPs with one or few reads – issue 1 of the Introduction section). Conversely, homozygous regions are much less prone to these confounding effects. Yet, missing data (issue 2), noisiness (issue 3) and sequencing errors (issue 4) can lower the power to identify homozygous diplotypes (*i.e.,* the combination of two gametic haplotypes). The general idea of the algorithm is, like in Hidden Markov Model (HMM), to use information of various SNPs around the imputed SNP, leaving unimputed the regions surrounding the recombination breakpoints laying between the two diplotypes. The locations of the recombination breakpoints are then inferred. Furthermore, instead of modeling error rates, we take an iterative approach to estimate them (Figure 1).

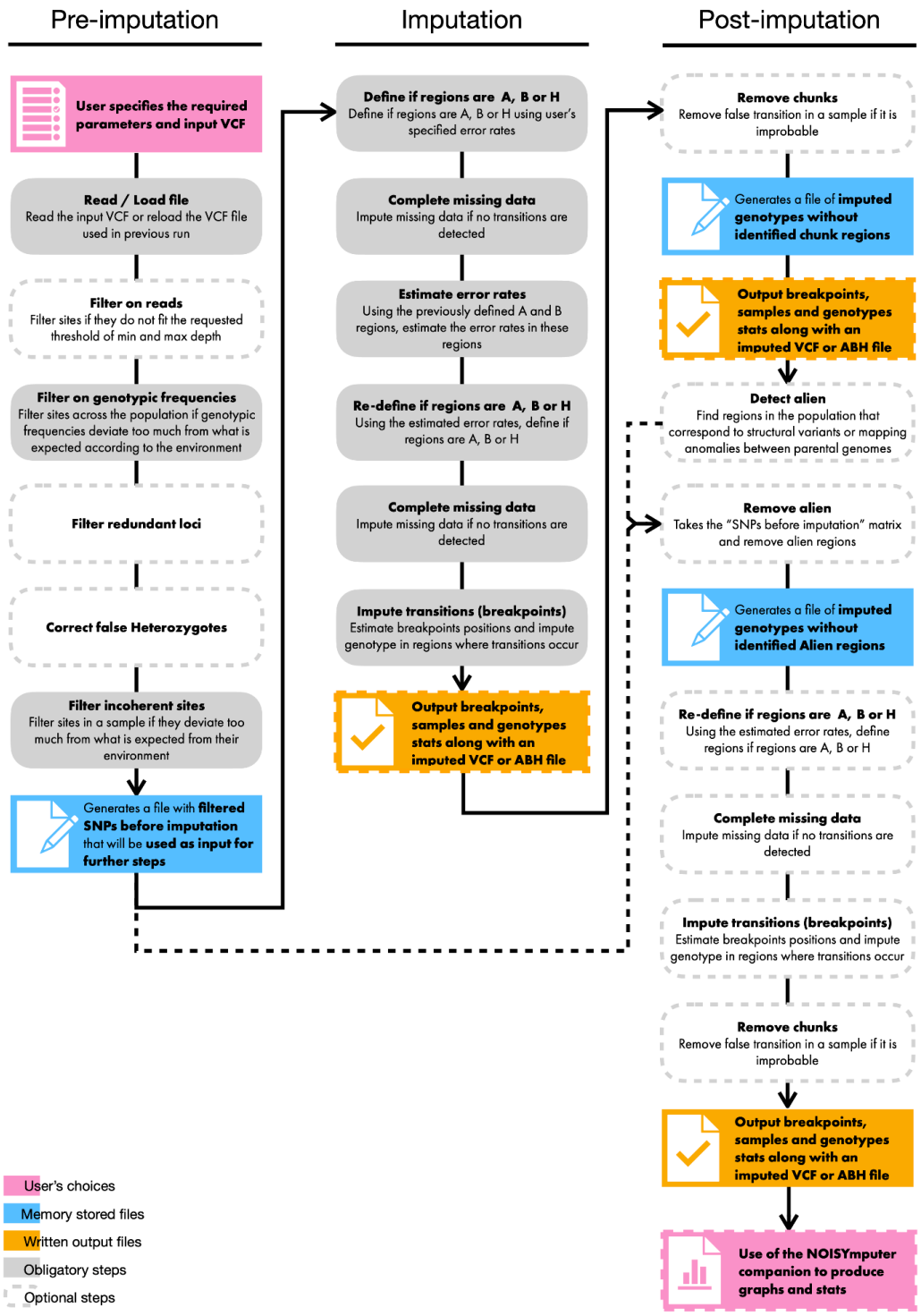

Figure 1. NOISYmputer's workflow. It is composed of three major phases: pre-imputation, imputation and post-imputation. Some steps are optional (dashed line) while others are required for the algorithm to complete.

Imputation - Step 1

Let's consider a chromosome of an F_2 individual with one single recombination breakpoint that separates a homozygous diplotype (AA; BB) from a heterozygous diplotype (AB, or BA, equivalent thereafter). Let's also consider a set of SNPs evenly dispersed on the physical genome, say, every 500 base pairs (bp). In the AA diplotype, and far from the breakpoint location, all SNPs should be genotyped as AA, except from the different kinds of errors cited above. To determine the genotype of a particular SNP, and due to these errors, one must consider not only its score in the VCF, but also its immediate "environment", that is, the SNPs that are located just before and just after it along the chromosome. Those surrounding SNPs help identify a potential error in the SNP scoring. Different approaches can be taken to look at the SNP environment. In segregating populations, the great majority of the genome is exempt from crossing overs. Indeed, when implementing a sliding window method like described hereby, the expected proportion of the genome with no recombination in the window is $P_{noXO} \approx 1 - \left(\frac{1}{100N}\right) D(8m - 2)$, where m is the number of SNPs in the sliding window, N is the total number of SNPs, and D is the expected genome size in centimorgans (cM). Hence, in almost the entire genome except the crossover regions there are only three possible genotypes, depending on the population type. Thus, instead of calculating all the likelihoods of possible paths (like in Hidden Markov Model methods), the problem is reduced to calculate the likelihoods of the data for the three possible diplotypes. Furthermore, there is no need to include transition (*i.e.*, recombination) probabilities. The main advantage of this approach is its computation time, which increases linearly according to the diplotype size, while it increases exponentially in the Hidden Markov Model process. We now describe the algorithm with the example of an F_2 population.

In practice, one defines starting values for error rates for reads A (e_A) and B (e_B), being respectively the probability of observing a B read (O_B) whereas the genotype is truly AA and observing an A read (O_A) whereas the genotype is truly a BB.

$$e_A = p\left(O_B|AA\right) \qquad e_B = p\left(O_A|BB\right)$$

We allow different error rates for A and B reads since the A and B parents are generally not equally (genetically) distant from the reference genome. For example, once could set $e_A = 0.005$ and $e_B = 0.003$ if Parent B is closer genetically to the Reference genome than Parent A is. Those values will be automatically refined after one or several rounds of imputation.

Thus, at homozygous sites, the probability of observing an A read if the true genotype is AA is

$$p\left(O_A|AA\right) = 1 - e_A$$

and the probability of observing a B read if the true genotype is BB is

$$p\left(O_B|BB\right) = 1 - e_B$$

At heterozygous (AB) sites, and assuming that the A and B reads have the same chance to occur, the probabilities of observing A and B reads are

$$p\left(O_A|AB\right) = \frac{1}{2}\, p(O_A|AA) + \frac{1}{2}\, p(O_A|BB) = \frac{1}{2}\,(1 - e_A) + \frac{1}{2}\, e_B$$

$$p\left(O_B|AB\right) = \frac{1}{2}\, p(O_B|BB) + \frac{1}{2}\, p(O_B|AA) = \frac{1}{2}\,(1 - e_B) + \frac{1}{2}\, e_A$$

Consider a chromosome (chr. for short) with n SNPs. For each site SNP_j of the chromosome, define a symmetrical window (W_j) containing the SNP_j at its center, m SNPs before it in the sequence and m SNPs after it (with read count > 0). SNPs that are located in chromosome ends are omitted, since it is not possible to define symmetrical windows around them. This case is discussed later on.

For each site SNP_i of the W_j window three situations are possible: i) the genotype G_i of the SNP_i is AA (homozygous for parent A allele), ii) the genotype G_i is BB (homozygous for parent B allele) or iii) the genotype G_i is AB (heterozygous).

By using the binomial distribution with sample size equal to n_i and the number of successes equal to nA_i (and thus of fails equal to nB_i), we estimate the likelihood of observing a given combination of reads (nA_i and nB_i as $n_i = nA_i + nB_i$) at SNP_i, knowing already the probability of observing A reads under the three possible genotypes:

$$P[nA_i \mid p(O_A \mid AA)] = \binom{n_i}{nA_i} p(O_A \mid AA)^{nA_i} (1 - p(O_A \mid AA)^{n_i - nA_i} = \binom{n_i}{nA_i} p(O_A \mid AA)^{nA_i} p(O_B \mid AA)^{nB_i}$$

$$P[nA_i \mid p(O_A \mid BB)] = \binom{n_i}{nA_i} p(O_A \mid BB)^{nA_i} (1 - p(O_A \mid BB)^{n_i - nA_i} = \binom{n_i}{nA_i} p(O_A \mid BB)^{nA_i} p(O_B \mid BB)^{nB_i}$$

$$P[nA_i \mid p(O_A \mid AB)] = \binom{n_i}{nA_i} p(O_A \mid AB)^{nA_i} (1 - p(O_A \mid AB)^{n_i - nA_i} = \binom{n_i}{nA_i} p(O_A \mid AB)^{nA_i} p(O_B \mid AB)^{nB_i}$$

Since the binomial factor is the same for the three possible genotypes, it can be omitted in the calculations. Then, individual relative probabilities that the genotype G_i of the SNP_i is AA, BB or AB are defined as:

$$p\left(G_i = X\right) = P[nA_i \mid p(O_A \mid X)] \,/\, \sum_X P[nA_i \mid p(O_A \mid X)]\,,\ with\ X = AA,\ BB,\ AB$$

The probabilities for the window's diplotype around the SNP_j to be AA, BB or AB are obtained by multiplying the individual probabilities of all the SNPs in the window. As multiplication of probabilities can result in very small numbers, we add their logarithms instead to avoid reaching the precision limit of the computer:

$$\rho_X = \sum_{i = SNP_j - m}^{SNP_j + m} log\left[p\left(G_i = X\right)\right],\ with\ X = AA,\ BB,\ AB$$

Finally, the relative probabilities for the window's W_j around the SNP_j to be AA, BB or AB are defined as:

$$P\left(W_j = AA\right) = exp\left(\rho_{AA}\right) / \left(exp\left(\rho_{AA}\right) + exp\left(\rho_{BB}\right) + exp\left(\rho_{AB}\right)\right)$$
$$P\left(W_j = BB\right) = exp\left(\rho_{BB}\right) / \left(exp\left(\rho_{AA}\right) + exp\left(\rho_{BB}\right) + exp\left(\rho_{AB}\right)\right)$$
$$P\left(W_j = AB\right) = exp\left(\rho_{AB}\right) / \left(exp\left(\rho_{AA}\right) + exp\left(\rho_{BB}\right) + exp\left(\rho_{AB}\right)\right)$$

A genotype is assigned to the SNP j if the relative probability of its surrounding window is superior to a given threshold α. . To guarantee that no SNP is falsely genotyped, the α threshold is set to a very stringent value (0.999 by default) . SNPs with $P\left(W_j\right) < \alpha$ for all genotypes are assigned a missing data value.

We repeat the process for each SNP j of the chromosome. For chromosome ends, the procedure is the similar except that the window m on the end side is smaller due to the lack of sites available to left or right of SNP_j.

This leaves two types of chromosomic regions unimputed and filled with missing data: 1) regions between imputed chromosome segments with identical diplotypes and for which none of the criteria are matched to assign a genotype, and 2) regions near recombination breakpoints.

Imputation - Step 2

Step 2 consists in filling the unimputed regions with the surrounding genotype, with the condition that they are surrounded (left and right) by identical imputed genotypes. This procedure assumes that a double recombination event is very unlikely. The maximum region size that is authorized for data filling can be calculated using the local recombination rate, which is calculated either from external data (e.g., another saturated map from the same species), or from the current data of the entire F_2 population, imputed from Step 1. So regions larger than the maximum size estimated with either method are left unimputed. It is desirable to use an interference model to estimate the distances (in cM), for instance the one implemented in the Kosambi mapping function (Kosambi 1944).

This step leaves the breakpoint regions unimputed.

We then estimate new values for e_A and e_B by comparing the observed data with the newly imputed regions.

This is done by simply counting the proportion of A reads in BB-imputed segments, and the proportion of B reads in AA-imputed segments.

Imputation - Step 3

Step 3 consists in imputing the SNP genotypes in the regions near the recombination breakpoints – between diplotypes of different states. The general idea is to determine an interval of high probability of presence (loose support interval) of the crossover, then to calculate the likelihood of the data under the hypothesis of a recombined segment.

This procedure allows determining with high confidence a loose support interval where the recombination breakpoint is located. Here we take the example of a segment BB to the left of the breakpoint and a segment AB to the right. Since we already know from Step 1 which are the two genotypes at the left and the right of the breakpoint, we only need to consider the only two possible diplotypes, BB and AB. This saves one degree of freedom.

If k defines the closest SNP position to the point where $p(W_j = BB) = p(W_j = AB)$ in Step 1, take $k - 2m$ and $k + 2m$ as starting points to guarantee that the breakpoint is covered by the interval. Then, for each SNP_j of the scanned area, recalculate $p(W_j = BB)$ and $p(W_j = AB)$, but this time in asymmetric windows of size m. For BB, define a window from SNP_j to $SNP_j + m$ and for AB a window from $SNP_j - m$ to SNP_j. And then, following calculations similar to Step 1 but omitting the probabilities for the AA genotype:

$$p(W_j = BB) = exp(\rho_{BB})/(exp(\rho_{BB}) + exp(\rho_{AB})) \text{ in the B window}$$

$$p(W_j = AB) = exp(\rho_{AB})/(exp(\rho_{BB}) + exp(\rho_{AB})) \text{ in the H window}$$

Starting from $k - 2m$, and progressing to the right, we look for the first site SNP_j for which :

$$P_{SI} = (1 - p(W_j = BB))(1 - p(W_j = AB)) > \alpha_{SI}, \text{ with } \alpha_{SI} = 0.05 \text{ by default.}$$

The breakpoint support interval is defined between the first position from the left (k_L) and from right (k_R) where $P_{SI} > \alpha_{SI}$.

The breakpoint support interval and position are then estimated inside the loose support interval. To do so, for each SNP_j in the breakpoint interval k_L to k_R, a probability P_{bkp} that the diplotype's window contains a breakpoint in its middle is estimated. Define a left window for $p_{bkp}(W_j = BB)$ that includes the SNP_j and goes to the left until the window reaches $m/2$ SNPs with at least a read (the left boundary of this window is called m_L) and a right window for $p_{bkp}(W_j = AB)$ that starts at $SNP_j + 1$ and goes to the right until the window

reaches $m/2$ SNPs with at least one read (the right boundary of this window is called m_R). Values of m_L and m_R are recalculated for each SNP_j.

The log-probabilities for the left and right segments are:

$$\rho_{bkp}\left(W_j = BB\right) = \sum_{i=m_L}^{j} log\left[p\left(G_i = BB\right)\right]$$

$$\rho_{bkp}\left(W_j = AB\right) = \sum_{i=j+1}^{m_R} log\left[p\left(G_i = AB\right)\right]$$

Then, the probability that the SNP_j and $SNP_j + 1$ are defining the breakpoint is:

$$p_{bkp}\left(BK_j\right) = exp\left(\rho_{bkp}\left(W_j = BB\right) + \rho_{bkp}\left(W_j = AB\right)\right)$$

And after normalization:

$$P_{bkp}\left(BK_j\right) = p_{bkp}(BK_j) \,/max\left(p_{bkp}\left(BK_z\right): z = k_L, \ldots, k_R\right)$$

The breakpoint is estimated in the middle of the interval defined by the SNP having the maximal $P_{bkp}(BK_j)$ and the next SNP to its right.

Finally, the unimputed genotypes in the breakpoint area are completed in assigning the BB genotype to the SNPs to the left of the SNP with the max $P_{bkp}(BK_j)$ (included) and AB to the right.

Imputation of breakpoint positions for the other types of homozygous-heterozygous transitions (AB→BB, AA→AB, AB→AA) are easily derived from the example beforehand.

The support interval for the breakpoint around its most likely position can be defined in searching for the SNPs (left and right starting from the SNP with the maximum $P_{bkp}(BK_j)$) for which $-\,log_{10}\left(P_{bkp}\left(BK_j\right)\right) \geq \alpha_{drop}$, where α_{drop} is the threshold of "drop" of the P_{bkp} value that the user chooses to accept (set to 1 by default).

Filtering options – before imputation

Genotypic frequencies, heterozygosity, missing data

The program can filter out SNPs for heterozygosity, percentage of missing data and parental genotypic frequencies. Min and max filtering values can be manually entered (usually not recommended though), or the program can calculate them from the genotype matrix imported from the VCF. In this case, genotypic frequencies are calculated for each SNP, and the filter values are derived from the extreme percentiles of the frequency distribution. Correction factors can be applied to the percentiles, to avoid too small or too large values.

Read counts

SNPs with too few or too many reads can be eliminated. This can be useful to, for instance, remove SNPs in duplicated regions.

Incoherent SNPs

In sequence-based genetic mapping, it is common to observe SNPs that do not segregate the same way as their immediate environment, indicating a probable mapping error due to, for instance, structural variation between the reference genome and the population parents, or between the parents, or both. As segregation distortion is a frequent phenomenon in many organisms, the Mendelian expected frequencies cannot be used to analyze the SNP segregation. Instead, the procedure defines a window of n SNPs around each tested locus. For each window/SNP couple, it calculates the genotypes AA, BB and AB frequencies and the reads A and B

frequencies across the population from the genotypes called in the VCF and compares the SNP and the window segregations using a chi-squared test, where expected counts are the observed frequencies in the window multiplied by the population size. It then filters out SNPs for which the chi-squared statistic exceeds a defined threshold for genotypes or reads frequencies.

Filtering options – after imputation

Incoherent chromosome segments (single individual)

Even after imputation and the different filtering operations, some few, improbable chromosome short diplotypes can still remain in the imputed matrix – we call them "small chunks". The procedure identifies each small chunk composed of identical alleles, embedded in a homogeneous genomic environment that has a different allele. The method resembles the one used in Imputation - Step2.

Consider two SNPs A and C that define the bounds of a region imputed as H and surrounded by regions imputed as A or B. Search for the SNP B that is the closest to the middle point between A and C (in cM). Also search for an SNP D before the SNP A so that $d_{DA} \sim d_{AB}$, and an SNP E *after* the SNP C so that $d_{CE} \sim d_{BC}$.

Then, calculate the recombination fractions r_{DB} and r_{BE} from d_{DB} and d_{BE} using the inverse of the Kosambi mapping function. Then the maximum probability of the "chunk" to be different to the surrounding genotype is

$$r_{ABC} = r_{DB} r_{BE}$$

The chunks for which $r_{ABC} \leq \alpha$ are restored with the surrounding genotype; α is set to 0.001 by default.

Incoherent chromosome segments (cross-population)

Entire chromosome segments can be misplaced due to genomic structural variation. Such segments are called "aliens" in the program. If their size is too large, the chi-square procedure that filters out the incoherent SNPs may fail to identify them since it is run *before* the imputation. Alien segments are easily detected, as they produce severe map expansion. The procedure analyzes rapid changes in the slope of the cumulated centimorgans of the genetic map calculated from the imputed matrix. When a marker is producing such change, the procedure searches for the next SNP that is closely linked to the SNP just before the slope change. It then eliminates all the SNPs that are in-between.

Running the program

Algorithm implementation

The program is implemented in Java, as a Spring Boot project. Spring Boot is an open-source Java framework used to create standalone java applications. The executable .jar is built using Maven, an open-source build tool.

Paths to datafiles and working folders paths, as well as parameters for imputation and filtering can be entered in a config file or directly in the command line. A NOISYmputerResults folder is automatically created, where the program writes all the output files.

Data specifications

In this current version, NOISYmputer is built and extensively tested to perform on F_2 intercross data, that is, the progeny from F_1 self-fertilization (F_2) but can also be used on recombinant inbred lines by single seed descent from the F_2 (SSD).

Input for NOISYmputer are standard Variant Call Format (VCF) files, with chromosome coordinates. Genotypes, alleles depth must be present in the VCFs. We consider the case of low coverage, that is, the sum of all sequences produced per sample is equivalent to 1-3 times (1-3 x) the size of the reference genome used. Ideally, the VCF should contain only bi-allelic single-nucleotide polymorphisms (SNPs), however NOISYmputer is automatically filtering sites that would not fit this requirement. We do not treat here the case of small indels.

Parental lines need to be included in the VCF file with the prefix "Parent" in their name. The VCF file can be compressed in the gzip ".gz" format.

Results

NOISYmputer, Tassel-FSFHap and LB-Impute were run on the IFB Core cluster (specs. available at https://ifb-elixirfr.gitlab.io/cluster/doc/cluster-desc/) with one allocated node per job and 32 GB to 64GB of RAM to be sure to not limit the efficiency of the software tested.

Details on parameters used for the three imputation methods are provided in Supplementary Data.

Using simulation to calibrate NOISYmputer

To test NOISYmputer's accuracy and precision in breakpoints estimation, we used simulated F2 datasets generated using PopSimul (https://forge.ird.fr/diade/recombination_landscape/popsimul). A set of 84 VCFs with n = 300 samples and different settings of marker density, mean depth and error rate were generated for a final expected map size of 180 cM (an average of 3-4 breakpoints per sample) to mimic the chromosome 1 of rice. Using five different impute half window sizes, we compared the outputs of NOISYmputer to the known positions of breakpoints in simulated data. In total, a set of 420 combinations have been analyzed. All combinations and tested parameters are listed in Table 1. The results of these analyses confirmed that NOISYmputer efficiently detects breakpoints and estimates precisely their positions.

Table 1. Different values used in PopSimul to generate simulated F2 VCFs with different marker density, mean depth and error rate. All possible combinations of these parameters have been tested and imputed using a range of imputation windows in NOISYmputer.

Parameters	Marker density (in number of markers along the chromosome)	Mean depth (in X)	Error rate	NOISYmputer impute half window size
Tested values	220,000 180,000 100,000 66,000	0.5 1 1.5 2 2.5 3 4	0.05 0.01 0.005	15 20 30 50 100

Detection power

We assessed the **NOISYmputer's ability to correctly detect all breakpoints within samples** by comparing positions of breakpoints found by NOISYmputer to those of simulated datasets. We consider a breakpoint correct when the simulated breakpoint position falls within NOISYmputer's loose support interval, along with the correct transition type.

Across all 420 VCFs, representing 455,000 breakpoints to detect, on average, NOISYmputer demonstrated robust detection power, correctly finding 99.5% of simulated breakpoints (median at 99.6%). NOISYmputer also displays a reliable accuracy as, on average, 98.9% of breakpoints identified correspond to actual breakpoints (with a median at 100%). Thus, NOISYmputer presents an overall excellent accuracy and power in detecting breakpoints.

To better understand the impact of each parameter and their interaction on NOISYmputer performance, we used a principal component analysis (PCA). Accuracy, primarily influenced by error rates, is also affected by imputation window size when excessively large. Conversely, smaller window sizes enhance detection power. Also, higher marker density correlates with improved detection power, as lower densities limit NOISYmputer's ability to identify breakpoints in regions with high recombination rates.

Some specific combinations decreased NOISYmputer detection accuracy and/or power but overall the lower performances were still convincing. For instance, the lower accuracy is of 72.3% (with error rates at 0.05 and smaller imputation window sizes of 15, and the lowest power is of 96.9% (with larger imputation window sizes of 100). This was expected as small windows with high levels of noise are prone to false positive breakpoints. On the other hand, large windows (especially if coupled with low depth or marker density) may miss double recombination events, leading to more false negatives (Figure 2 A and B).

Some characteristics of VCF files depend on the model species or sequencing type, such as depth or marker density or species model, and are not under the user's control. Knowing this, we looked for the imputation window size giving the best results in both breakpoint detection accuracy and power with the closest VCF to real F2 rice data we have. In both cases, the optimal results are obtained by the imputation half window size of 30. Thus, we'll use this later on when exposing NOISYmputer to real datasets.

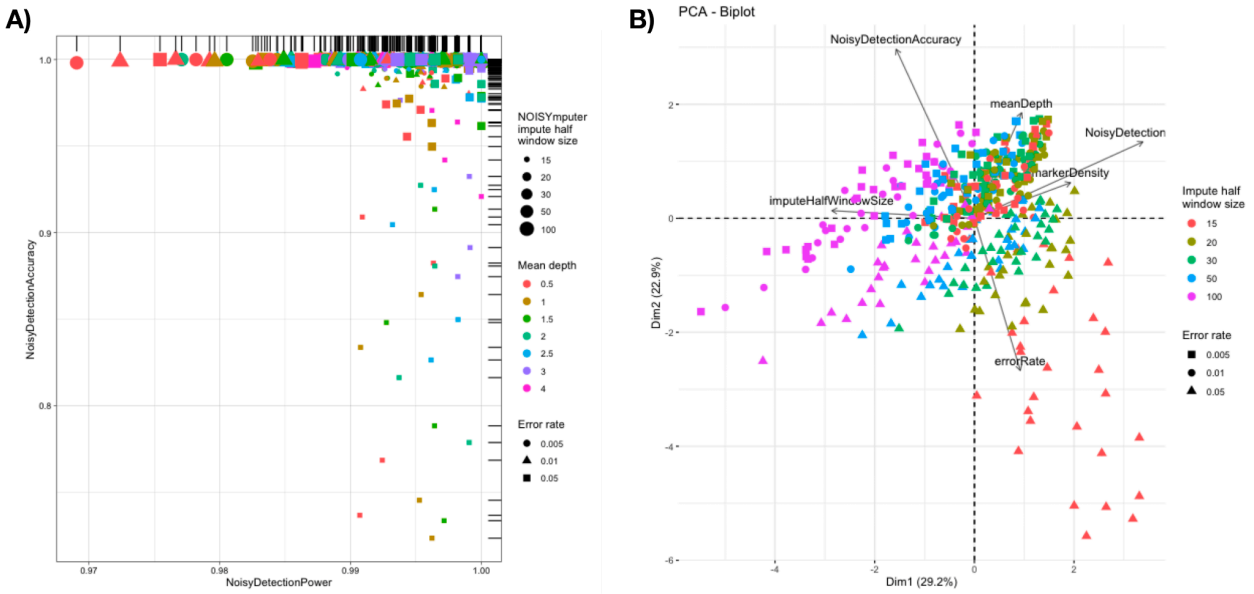

Figure 2. Parameters and VCFs characteristics impacting the most NOISYmputer results based on 420 VCFs analyzed. **A)** A representation of the NOISYmputer detection accuracy (proportion of NOISYmputer breakpoints being actual breakpoints from simulated data) in function of NOISYmputer detection power (proportion of simulated breakpoints correctly found by NOISYmputer). NOISYmputer is showing excellent power detection and accuracy with at least 72.3% and 96.9% respectively. **B)** PCA Biplot of NOISYmputer showing VCFs characteristics and imputation window size influencing detection accuracy and precision with simulated VCFs. The lowest detection powers are observed when high error rates are coupled with a small half impute window size in NOISYmputer; The lowest accuracies correspond to VCF imputed with a large half impute window size in NOISYmputer and can be accentuated by very low depth (≤ 1X) and/or marker density (< 66 000 sites / 44 Mb).

Precision success

Noisymputer's precision was estimated by computing the difference between the simulated breakpoints positions and the estimated one by NOISYmputer. We took into account the size of the support interval and its marker density to estimate the number of SNPs discrepancy with the actual breakpoint position.

Across all 420 VCFs, a difference of 1427 bp on average (equivalent to a discrepancy of 2 SNPs) was observed. However, the median difference is lower, with only 245 bp (< 1 SNPs discrepancy). This disparity between the median and mean is mainly due to extreme combinations, particularly low depth combined with high error rate. Notably, variance is higher in 0.5X coverage VCFs, becoming more homogeneous at 1X coverage.

Regarding the imputation window, smaller half windows result in lower average differences between NOISYmputer and simulated positions but increase the median difference. Consequently, smaller windows enhance overall precision while potentially increasing the occurrence of extreme discrepancies.

Error rate estimations

Regarding error rates (e_A and e_B), they are recalculated after a first round of imputation step 1. We do obtain a

100% correct estimation of the error rate with NOISYmputer based on known error rate of simulated VCFs (Supplementary Data 1).

Confirmed efficiency on real data and comparison with other methods

We assessed the performance of our software on two real datasets: i) a Maize F2 population in GBS with 91 samples, including parents, and ii) Rice F2 population with 3X coverage in whole-genome sequencing (WGS) comprising 222 samples, including 30X parents. Details of how real datasets were generated are summarized in Supplementary Table S1 of the Supplementary Data.

In real data, direct estimation of imputation accuracy may be challenging due to the unknown true state at each locus. However, we can assess imputation quality indirectly by comparing the final genetic map to existing high-quality maps. A correctly imputed dataset should yield a map size consistent with those derived from high-quality marker data. Conversely, datasets with a high rate of genotyping errors will exhibit map expansion, resulting in a longer genetic map – in centimorgans (cM) – due to falsely imputed recombination breakpoints.

Using map size estimates in centimorgans (cM) of chromosome 1 of these datasets, we compared the results of NOISYmputer to LB-Impute and Tassel FSFhap (Figure 3 and Table 2). Concerning the maize GBS dataset, LB-Impute and Tassel FSFhap strongly overestimates the map size expected from high-quality datasets (respectively 633 cM and 13 271 cM), whereas NOISYmputer is in range with the expected map size (203 cM). Regarding the Rice WGS dataset, while both LB-Impute and Tassel FSFhap yielded maps larger than expected (over 23 436 cM and 337 750 cM respectively), NOISYmputer estimated again a size close to the expected value (213 cM). It's important to note, however, that LB-Impute does not output a proper estimate of breakpoint positions and thus the computed genetic map is highly underestimated (see supplementary materials for more details). Also, results from TASSEL on PopSimul data are highly impacted by very large map size estimations for high (5%) error rates VCF, while they are qualitatively similar to NOISYmputer's for lower (1% and .5%) error rates.

Table 2. Comparison of estimated map size to expected map size on three different datasets using NOISYmputer, TASSEL FSFHap and LB-Impute. The 84 VCFs generated using PopSimul can have a number of markers of about 66 000, 100 000, 180 000 or 220 000 markers depending on the settings used to generate the VCFs. Overall, NOISYmputer is showing a considerably higher accuracy in map size estimation compared to other tested softwares.

Dataset	Software	Estimated map size (cM)	Expected map size (cM)	Initial number of markers in VCF
F2 Maize GBS n = 91 samples including parents	**NOISYmputer**	203	~200	17945
	TASSEL FSFHap	13 271		
	LB-Impute	633		
F2 Rice WGS n = 222 samples including 30X parents	**NOISYmputer**	213	183	254095
	TASSEL FSFHap	337 750		
	LB-Impute	23 436		
84 PopSimul VCFs n = 300 samples in each including parents	**NOISYmputer**	180	180	Different number of marker depending on the VCF settings
	TASSEL FSFHap	479 399		
	LB-Impute	N/A		

To further estimate the performance of NOISYmputer on real datasets we also performed comparisons on breakpoint detection accuracy, power and position estimate among the 222 F2 Rice population. This dataset includes 20 samples artificially subsetted to 3X (that we call pseudo-3X). These 20 samples allow for a more robust evaluation as their breakpoints are well estimated thanks to their better depth. We processed similarly to the simulated analyses and compared breakpoint detection specificity, power, and precision of breakpoint estimates for NOISYmputer against the accurately estimated breakpoints at 20X coverage. Unfortunately, we were not able to compare NOISYmputer results to those of TASSEL FSFHap and LB-impute as, even if we managed to retrieve each breakpoints position estimate, we could not easily check which were actual breakpoints and which were false positives. Overall, NOISYmputer demonstrated excellent results with, on average, 99% accuracy and 97% detection power. Concerning precision, on average the difference is of 10,219 bp precision (median: 415 bp), equivalent to 2 SNPs discrepancy on average (median: 1).

A resource-optimized software, cost-effective in CPU and RAM.

In our comparative analysis of the new software against established counterparts, we conducted comprehensive benchmarks focusing on execution time and RAM usage (Table 3 and Figure 3). To do so, we ran NOISYmputer, TASSEL FSFHap and LB-Impute on simulated and real datasets. We then retrieved their CPU time, "wall clock" execution timDifferences in map size are mostly due to misinterpretation of noisy data by Tassel FSFHap and LB-Impute. These discrepancies frequently arise in regions corresponding to structural variations between parental genomes. Such variations can occur when attempting to map onto regions found exclusively in the Parent A genome, which serves as the reference. In such cases, mapping reads from Parent B poses challenges due to the absence of "B reads", induced by deletions in the Parent B genome, causing these regions to resemble an A haplotype and resulting in false recombination events according to imputation softwares. The phenomenon is accentuated in WGS dataset compared to GBS dataset as the complete genome is sequenced and map and thus the number of markers increases. Including more sites, inducing sites belonging to peculiar genomic structures, can hinder the quality of imputation if the software does not take into account the coherence of a marker with its surrounding environment in the sample, along with its coherence with the entire population at this locus. Though Tassel FSFHap and LB-impute might be precise in the estimated breakpoints positions, their lack of accuracy in breakpoints detection leads to results, on whole genome datasets, hard to use without any heavy filtering steps. NOISYmputer on the other hand is very efficient at dealing with artifact mapping issues or divergence between parental genome structures.

e and RAM utilization using the *seff* command on the cluster where they were executed.

Concerning the **F2 Maize GBS dataset**, NOISYmputer is going about 10 to 45 times faster than TASSEL FSFhap and LB-Impute respectively. It also uses less RAM as NOISYmputer requires about 3.4 GB of RAM which is 3 times less than TASSEL FSFHap and 21 times less than LB-Impute.

When comparing the ressources usage on the **F2 Rice WGS dataset,** NOISYmputer is considerably faster and more efficient in RAM usage than the two other softwares. NOISYmputer uses slightly less RAM than TASSEL FSFHap and is 13 times faster (< 6 min vs. 1h19). LB-Impute however is demonstrating CPU and RAM efficiency way below the other two as NOISYmputer is using 9 times less RAM and goes 145 times faster.

Due to the long computation time on this single smaller dataset, LB-Impute was excluded from the remaining comparisons with the 84 PopSimul VCFs with 300 samples. It's interesting to note though that TASSEL FSFHap resource efficiency is better **on simulated datasets** even though they have more samples. Indeed, TASSEL FSFHap is using on average 1.93 GB of RAM, whereas NOISYmputer is stable at 3.61 GB. NOISYmputer is still a bit faster than TASSEL FSFHap on average, with ~ 5 min, while TASSEL FSFHap ran ~ 9 min. This underlies the problem with TASSEL FSFHap and noisy data, partly due to structural variants and noisy datasets. These results underscore NOISYmputer's efficiency improvement in processing imputation tasks, especially compared to what is available currently to impute bi-parental populations. .

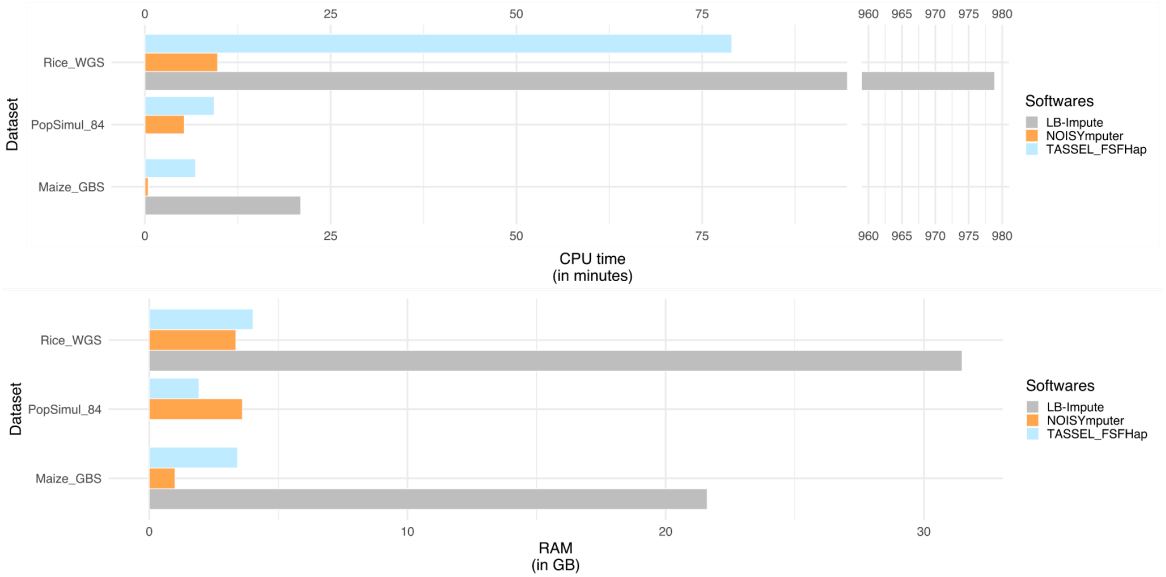

Figure 3. Barplot of the CPU time and RAM resource usage from NOISYmputer (orange), LB-Impute (gray) and TASSEL FSFHap (blue) on three datasets. Rice_WGS is a F2 Rice WGS dataset with n=222 samples including parents; PopSimul_84 values are the result of used resources, on average, across 84 VCFs generated with PopSimul, each VCF containing n=300 samples including parents, and having a range of depth, marker density and error rate to mimic different characteristics of F2 VCFs; Maize_GBS is a F2 Maize GBS dataset with n=91 samples including parents (with a lower marker density than Rice_WGS). NOISYmputer is overly fast and RAM efficient in all conditions, doing better than other softwares with an exception for RAM usage on simulated VCF files of PopSimul. No gray bars are present on PopSimul_84 as LB-Impute has not been benchmarked for resource usage on the 84 simulated VCFS.

Table 3. CPU and RAM usage of NOISYmputer, Tassel FSFHap and LB-Impute for three datasets based on the output of the *seff* command on the IFB cluster. NOISYmputer 1st and 2nd Run are displayed as NOISYmputer can show even better CPU time usage at second run due to the raw file being already stored. For LB-Impute, as imputation is processed in two steps, CPU time and execution time results are the sum of the two steps; RAM usage corresponds to the higher RAM usage of the two steps (offspring imputation). For the 84 PopSimul VCFs section, results correspond to the mean of ressources usage for each of the 84 PopSimul VCFs for a half imputation window of 30 with NOISYmputer and the default window size (50) of TASSEL FSFHap. *LB-impute does not properly compute a breakpoint position and is heavily time and ram consuming on 222 samples WGS. Thus, we did not benchmark the 84 PopSimul VCFs with LB-impute. All tests have been conducted on the IFB Core cluster.

Datasets	Software		CPU time (h:m:s)	Total execution time (h:m:s)	RAM (GB)
F2 Maize GBS N = 91 samples including parents	**NOISYmputer**	2nd Run	00:00:07	00:00:09	1.00
		1st Run	00:00:27	00:00:28	1.00
	TASSEL		00:06:49	00:04:35	3.42
	LB-Impute		00:20:59	00:21:04	21.61
F2 Rice WGS n = 222 samples including 30X parents	**NOISYmputer**	2nd Run	00:06:49	00:04:35	3.42
		1st Run	00:09:47	00:06:44	3.36
	TASSEL		01:19:00	01:19:06	4.02
	LB-Impute		16:18:52	16:19:21	31.48
84 PopSimul VCFs with n = 300 samples each including parents	**NOISYmputer**	1st Run	00:05:18	00:05:39	3.61
	TASSEL		00:09:20	00:09:24	1.93
	LB-Impute*		NA	NA	NA

Availability and Future Directions

Availability

NOISYmputer is available as a multiplatform (Linux, macOS, Windows) Java executable at the URL https://gitlab.cirad.fr/noisymputer/noisymputerstandalone/-/tree/1.0.0-RELEASE?ref_type=tags. The source code and the documentation is available at the same URL. A quarto markdown companion (compatible with R markdown and Jupyter notebooks IDE) that allow displaying graphics of statistics (genotypic frequencies on SNPs and samples) and graphical genotypes from NOISYmputer output files has been developed and is also available.

NOISYmputer and companion are distributed under the GNU Affero General Public License V3.0.

Future directions

NOISYmputer strength

Although previous methods have made significant advances in addressing the challenges listed above, the noisiness of imputed datasets are still evident by expanded genetic maps, excess heterozygosity, and probabilistically unlikely recombination events contained within a short physical interval. Here, we introduce an algorithm that, in a series of steps, addresses each source of error to create higher-quality datasets for improved trait mapping and genomics-assisted breeding. Our algorithm represents a step to systematically address all sources of NGS genotyping error, and hopefully the corrections made here will be recapitulated in future algorithm development. Indeed, key features of NOISYmputer are its pre- and post-filtering steps that other currently available softwares do not have. In filtering SNPs and segments that are incoherent with their environment and to the population local recombination landscape, NOISYmputer efficiently eliminates errors of genotype calling sequencing error or structural variants. These features truly make a difference when comparing NOISYmputer to Tassel FSFHap (Swarts *et al.* 2014) on real datasets for instance.

NOISYmputer is a resource-effective software developed in Java, allowing its integration in bioinformatics pipelines. NOISYmputer is parallelizing computation at the sample levels when possible in the algorithm which increases its speed compared to existing softwares. The use of a Java standalone also allows to simulate parallelization in running each chromosome on a separate core of a server/cluster. Moreover, NOISYmputer employs a maximum likelihood method, instead of hidden Markov models, which considerably reduces computational efforts, compared to TASSEL FSFHap (Swarts *et al.* 2014) and LB-impute(Fragoso *et al.* 2016), while enhancing result accuracy and flexibility across diverse datasets. In fact, NOISYmputer is less sensitive to noisy regions (due to mapping artifacts for example) as it can handle large windows without being greedy in RAM and computation time to overpass complex regions.

The software's speed enables iterative refinement of parameter settings. For example, the size m of the window (in number of SNPs), like in other imputation programs (e.g., Tassel-FSFhap, LB-Impute), is arbitrarily fixed by the user. The most appropriate value for m depends on several factors, including depth and SNP density. A convenient way to determine which value for m to use is to run the imputation several times with different values until reaching the expected distribution of the number of recombination breakpoints per sample across the population (if previously known). Often, saturated genetic maps generated with other types of markers are available in the literature, from which the expected distribution is easily derived. With our rice data, the imputation algorithm gave best results with m equal to 30, so even a few runs should provide a satisfying window size.

Furthermore, NOISYmputer generates a .json file during the initial run, that is re-read for following runs, eliminating redundant tasks of reading or loading the input VCF each time, thus enhancing speed for subsequent launches on the same dataset.

Its robust performance extends to various VCF characteristics, accommodating differences in SNP quality, marker density, error rates, and sequencing depths. It's partly due to it being less sensitive to the SNP calling step used to generate the input VCF as NOISYmputer is estimating the probability of genotype using the allele

depth at each site, along with information of the surrounding environment and of the whole population. This results in NOISYmputer maintaining an overall good accuracy, power and precision on breakpoints even with very low coverage datasets (≤1X). Users should exercise caution however in selecting an appropriate imputation window size to mitigate the risk of false positives and negatives.

In addition to its performance benefits, NOISYmputer provides users with several comprehensive breakpoint confidence information allowing to further filter the identified breakpoints. This is a feature that is innovative and useful and not available in other softwares, to our knowledge. NOISYmputer also outputs statistics on genotypic/allelic frequencies, samples and genetic map among others.

Suggestions for Improvement

NOISYmputer could benefit from some improvements. The first one is including more population types. In its next version, we would like NOISYmputer to be able to handle F_2 backcross, or BC_1F_1, the progeny of the F_1 hybrid crossed with one of the parents (BC_1) ; doubled haploid of F_1 gametes (DH) ; F_2 intercross, that is, the progeny from F_1 self-fertilization (F_2); Recombinant inbred lines by single seed descent from the F_2 (SSD) or the BC_1F_1 (BCSSD); The unconventional mating design (UMD) BC_1F_3, derived by two generations of self-fertilization of BC_1F_1 individuals. For now, it has been extensively tested and optimized for F2 crosses between distant parents which might be one of the hardest profiles to estimate breakpoints from. We thus are confident that it could be adapted to these other types of crosses.

Breakpoint detection and accuracy could benefit from a different model. Currently, we test for the existence of a single transition inside the loose support interval in step 3. Testing for one, two or even three transitions in a single interval could increase the probability of finding close double recombination events if they happened to have a higher probability in the tested region. Breakpoint position estimation, on the other hand, might be improved by using the combinations of the NOISYmputer actual algorithm to a hidden markov model occuring in the step 3 of imputation only. This way, the window size at this step can be small and the region to scan is a constrained percentage of the genome only. It could increase the precision of the breakpoint position even more, without being too greedy in computation time.

NOISYmputer is robust on a broad range of samples and computation time is not an issue. However, part of the success of NOISYmputer lies in the fact it has pre-imputation filtering steps where it removes, among other things, incoherent SNP in a sample, meaning a SNP that does not segregate the same way as its immediate environment (often indicating mapping errors). This filtering incoherent step uses a Chi-square to evaluate if the observed pattern is reasonable. Unfortunately, Chi-square thresholds are dependent on the sample size analyzed. Thus, if a VCF of let's say 2000 samples was to be imputed with NOISYmputer, the user should be careful with the Chi-square threshold used. A solution to this issue would be to use a "Kramer's V" statistic instead (), which would be unrelated to the sample number in the VCF.

Finally, as we saw in the results section, the imputation half window size can have an impact on the outputs of NOISYmputer. We think NOISYmputer would benefit from an iteration process that would check different window sizes and analyze the convergence of the results to select the appropriate window size and thus have the best compromise between detection accuracy and power, along with precision.

Acknowledgements

Karine Labadie (CEA, Institut de Génomique, Genoscope, Evry, France) for sharing the WGS data for the sequencing of rice populations, Christine Tranchant-Dubreuil (IRD, Montpellier, France) for her help with retrieving the Rice_WGS data and François Sabot (IRD, Montpellier, France) for coordinating the IRIGIN project, Fabian Tobar (Universidad Javeriana, Cali, Colombia) for his supervision role of CS's Ph.D. We thank the Yale Center for Research Computing for guidance and use of the research computing infrastructure. The following programs supported parts of this initiative: the French Government France Génomique program through its International RIce Genome INitiative (IRIGIN) project, the RICE CGIAR Research Program and the French ANR project "LANDSREC" (ANR-21-CE20-0012-03).

We are grateful to the Institut Français de Bioinformatique for providing computing resources.

Author's contributions

AB participated in the algorithm development and implemented it in Java, took part in the manuscript conception. CT participated to the algorithm development, designed and ran the bioinformatics pipeline to call SNPs for the WGS dataset (F2 and SSD), benchmarked and compared all softwares, wrote the quarto markdown companion, tested the program for debugging and took part in the manuscript conception. AG designed and ran the bioinformatics pipeline to call SNPs for the Rice_WGS dataset for the previous version of NOISYmputer. CF ran LB-Impute for the Rice_WGS dataset on the Yale cluster and edited the manuscript. JFR designed the pipeline for the Sorghum_GBS dataset and ran the early tests of ABHgenotypeR. ML conceptualized the initial imputation algorithm for NOISYmputer, participated in its further development and took part in the manuscript conception.

References

Browning S. R., and B. L. Browning, 2007 Rapid and Accurate Haplotype Phasing and Missing-Data Inference for Whole-Genome Association Studies By Use of Localized Haplotype Clustering. The American Journal of Human Genetics 81: 1084–1097. https://doi.org/10.1086/521987

Browning B. L., X. Tian, Y. Zhou, and S. R. Browning, 2021 Fast two-stage phasing of large-scale sequence data. The American Journal of Human Genetics 108: 1880–1890. https://doi.org/10.1016/j.ajhg.2021.08.005

Davey J. W., and M. L. Blaxter, 2010 RADSeq: next-generation population genetics. Briefings in Functional Genomics 9: 416–423. https://doi.org/10.1093/bfgp/elq031

Elshire R. J., J. C. Glaubitz, Q. Sun, J. A. Poland, K. Kawamoto, *et al.*, 2011 A Robust, Simple Genotyping-by-Sequencing (GBS) Approach for High Diversity Species, (L. Orban, Ed.). PLoS ONE 6: e19379. https://doi.org/10.1371/journal.pone.0019379

Fragoso C. A., C. Heffelfinger, H. Zhao, and S. L. Dellaporta, 2016 Imputing Genotypes in Biallelic Populations from Low-Coverage Sequence Data. Genetics 202: 487–495. https://doi.org/10.1534/genetics.115.182071

Heffelfinger C., C. A. Fragoso, M. A. Moreno, J. D. Overton, J. P. Mottinger, *et al.*, 2014 Flexible and scalable genotyping-by-sequencing strategies for population studies. BMC Genomics 15: 979. https://doi.org/10.1186/1471-2164-15-979

Howie B., C. Fuchsberger, M. Stephens, J. Marchini, and G. R. Abecasis, 2012 Fast and accurate genotype imputation in genome-wide association studies through pre-phasing. Nat Genet 44: 955–959. https://doi.org/10.1038/ng.2354

Huang X., Q. Feng, Q. Qian, Q. Zhao, L. Wang, *et al.*, 2009 High-throughput genotyping by whole-genome

resequencing. Genome Res. 19: 1068–1076. https://doi.org/10.1101/gr.089516.108

Kosambi D. D., 1944 The Estimation of Map Distances from Recombination Values, pp. 125–130 in *D.D. Kosambi: Selected Works in Mathematics and Statistics*, edited by Ramaswamy R. Springer India, New Delhi.

Swarts K., H. Li, J. A. Romero Navarro, D. An, M. C. Romay, *et al.*, 2014 Novel Methods to Optimize Genotypic Imputation for Low-Coverage, Next-Generation Sequence Data in Crop Plants. The Plant Genome 7: plantgenome2014.05.0023. https://doi.org/10.3835/plantgenome2014.05.0023

Xu C., Y. Ren, Y. Jian, Z. Guo, Y. Zhang, *et al.*, 2017 Development of a maize 55 K SNP array with improved genome coverage for molecular breeding. Mol Breeding 37: 20. https://doi.org/10.1007/s11032-017-0622-z