Supplement of A Fast Data-Driven Method for Genotype Imputation, Phasing, and Local Ancestry Inference: MendelImpute.jl

June 28, 2021

1 Supplemental Material

1.1 Imputation Quality Scores

Consider the observed genotype $x_{ij} \in [0,2] \cup \{missing\}$ at SNP i of sample j and the corresponding imputed genotype g_{ij} derived from the two extended haplotypes of j. If S_i denotes the set of individuals with observed genotypes at the SNP, then `MendelImpute`'s quality score q_i for the SNP is defined as

$$q_i \quad = \quad 1 - \frac{1}{|S_i|} \sum_{j \in S_i} \left(\frac{x_{ij} - g_{ij}}{2} \right)^2.$$

Note that $0 \leq q_i \leq 1$ and that the larger the quality score, the more confidence in the imputed values. Because q_i can only be computed for the typed SNPs, an untyped SNP is assigned the average of the quality scores for its two closest flanking typed SNPs. Figure 1A plots each SNP's quality score in the 1000G Chr20 experiment summarized in Table **??**. For each sample, one can also compute the mean least squares error over all p SNPs to obtain a per-sample quality score. This is shown in Figure 1B. By default `MendelImpute` outputs both quality scores. Thus, investigators can perform post-imputation quality control by SNPs and by samples separately.

Empirically, it is rather common for a sample subject to harbor a few poorly imputed windows. Thus, we observe a long left tail in the histogram for per-sample error in Figure 1. Unfortunately, the bad windows generally do not exhibit any discernible regional patterns across subjects. We suspect that poorly imputed windows involve breakpoints that occur near the middle of a window. It is possible to detect such breakpoints by IBS matching and ancestry weighting [4]. We plan a detailed analysis of this issue in future work.

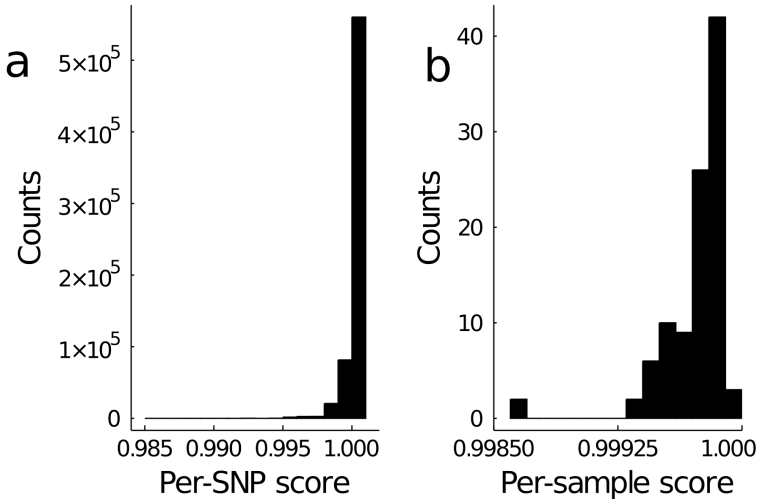

Figure 1: Histograms of per-SNP and per-sample quality scores for chromosome 20 in our 1000G analysis. By default `MendelImpute` computes (a) per-SNP quality scores and (b) per-sample quality scores. SNPs and samples with noticeably lower quality scores should be removed from downstream analysis.

1.2 JLSO Compressed Reference Haplotype Panels

The `jlso` format is constructed in three steps: (a) specify window intervals, (b) compute unique haplotypes in addition to hash maps to reference haplotypes in each window, and (c) save the result in a binary compressed format via the `JLSO.jl` package [1]. The resulting `jlso` files are 30-50x faster to read and 3-5x smaller in file size (varies depending on window width) than compressed VCF files in the `vcf.gz` format. Note the `jlso` format is simply a container object that facilitates reading and transferring large VCF files stored as Julia variables. In principle, all files that are slow to read can be pre-processed and stored in this alternative format for quicker access. As such, we similarly store ultra-compressed sample haplotypes, as discussed in Section 3.4, using the `JLSO.jl` package.

1.2.1 Adaptive Window Widths via Recursive Bisection

The first step in generating JLSO haplotype panel is to specify genomic window ranges. The width of genomic windows is an important parameter determining both imputation efficiency and accuracy. Empirically, larger window widths give better error rates but also increase the computational burden of the matrix multiplications and minimum entry search described in Section 2.2. The magnitudes of these burdens depend on local haplotype diversity. Thus, we choose window widths dynamically. This goal is achieved by a bisection strategy. After aligning all typed SNPs with the reference panel, initially we view all typed SNPs on a large section of a chromosome as belonging to a single window. We then divide the window into equal halves if it possesses too many unique haplotypes. Each half is further bisected and so forth recursively until every window contains

fewer than a predetermined number of unique haplotypes. Empirically, choosing the maximum number d_{max} of unique haplotypes per window to be 1000 works well for both real and simulated data. When a larger number is preferred, we resort to a stepwise search heuristic for minimizing criterion (**??**) that scales linearly in the number of unique haplotypes d. This heuristic is described above.

1.2.2 Elimination of Redundant Haplotypes by Hashing

Within a small genomic window of the reference panel, multiple haplotype pairs may be identical at the typed SNPs. Only the unique haplotypes play a role in matching reference haplotypes to sample genotypes. `MendelImpute` identifies redundant haplotypes by hashing. For each reference haplotype limited to the window, hashing stores an integer representation of the haplotype via a hash function. This integer serves as an index (key) to locate the reference haplotype (value). Put another way, hashing stores the inverse images of the map from reference haplotypes to unique haplotypes. In our software, the `GroupSlices.jl` package [2] identifies a unique key for each haplotype.

1.2.3 Save in binary compressed format

Since haplotypes are long binary vectors, the entire haplotype reference panel can be compactly represented using a single bit per entry. We have already divided the full reference panel into non-overlapping windows of various widths after proper alignment of all typed and reference SNPs in the window. For each window we save two compressed mini-panels. The first houses the unique haplotypes determined by just the typed SNPs. The second houses the unique haplotypes determined by all SNPs in the window, typed or untyped. The former is much smaller than the later, but each entry of both can be compactly represented by a single bit per entry in memory. Thus, for each window we save two compressed windows in addition to meta information and pointers that coordinate reference haplotypes with the two mini-panels per each window. This whole ensemble is stored in the jlso file. Because there are only a limited number of SNP array chips on the market, one can in principle store just a few jlso files on a universal source such as the cloud. The same JLSO compressed panel can also be re-used as long as the set of typed SNPs does not change drastically between different GWAS chips.

1.3 Parallel Computing and Memory Requirements

`MendelImpute` employs a shared-memory parallel computing model where each available core handles an independent component of the entire problem. Work is assigned via Julia's multi-threading functionality. When computing the optimal haplotype pairs in equation 1 of section 2.2, we parallelize over windows. This requires allocating c copies of $\mathbf{X}^T\mathbf{H}$ and $\mathbf{H}^T\mathbf{H}$, where c is the number of CPU cores available. Note the

dimensions of these matrices vary across windows. To avoid accruing memory allocations, we pre-allocate c copies of $n \times d_{max}$ and $d_{max} \times d_{max}$ matrices and re-use their top-left corners in windows with $d < d_{max}$. For intersecting adjacent reference haplotype sets (phasing), we parallelize over samples. This step requires no additional memory. Writing to output is also trivially parallelizable by assigning each thread to write a different portion of the imputed matrix to a different file, then concatenating these files into a single output file. Data import is not parallelized. Beyond allocating $\mathbf{X}^T\mathbf{H}$ and $\mathbf{H}^T\mathbf{H}$, our software requires enough memory (RAM) to load the target genotype matrix and the compressed haplotype reference panel.

1.4 Bias Correction for Initializing Missing Data

Since BLAS requires complete data, we must first initialize the missing data in each genotype vector $\mathbf{x}$ before computing $\mathbf{M}$ and $\mathbf{N}$ in equation 2 of section 2.2. This may introduce bias in our minimization of equation 1 if there is a high fraction of missing genotypes in the typed SNPs, for example above 10%. One way to alleviate bias is to initialize missing data with the mean and save all unique haplotype pairs minimizing criterion equation 1 under this convention. Once this set of optimal haplotype pairs are identified, we re-minimize criterion equation 1 but now skipping the missing entries of $\mathbf{x}$. That is equivalent to setting $x_k - h_{ik} - h_{jk} = 0$ when x_k is missing.

1.5 Avoidance of Global Searches for Optimal Haplotype Pairs

Recall that minimizing equation 1 of section 2.2 requires searching through all lower-triangular entries of the $d \times d$ matrix $\mathbf{M} + \mathbf{N}$, where d denotes the number of unique haplotypes in the window. When $d < 1000$, searching through all $\binom{d}{2} + d$ lower-triangular entries of $\mathbf{M} + \mathbf{N}$ via `MendelImpute`'s standard procedure is fast, but this global search quickly degrades as $d \to \infty$. Below we outline two heuristic procedures for large d. These heuristics typically produce sub-optimal solutions compared to global searches, so they should be used with caution.

1.5.1 Stepwise Search Heuristics

Consider minimizing the loss $f_i(\boldsymbol{\beta}) = \frac{1}{2}||\mathbf{x}_i - \mathbf{H}\boldsymbol{\beta}||_2^2$, where the d columns of $\mathbf{H} \in \{0,1\}^{p \times d}$ store unique haplotypes, p is the window width, and $\mathbf{x}_i$ is a sample genotype vector. The original problem minimizes $f_i(\boldsymbol{\beta})$ under the constraint that exactly two $\beta_j = 1$ and the remaining $\beta_k = 0$ or the constraint that exactly one $\beta_j = 2$ and the remaining $\beta_k = 0$. As an approximate alternative, one first finds the r unique haplotypes with the largest influence on $f_i(\boldsymbol{\beta})$. This is accomplished by identifying the r most negative components of the gradient

$$\nabla f_i(\boldsymbol{\beta}) = -\mathbf{H}^T(\mathbf{x}_i - \mathbf{H}\boldsymbol{\beta}) = -\mathbf{H}^T\mathbf{x}_i + \mathbf{H}^T\mathbf{H}\boldsymbol{\beta}$$

at $\beta = \mathbf{0}$. These are the r directions of steepest descent. Note that $\nabla f_i(\mathbf{0}) = -\mathbf{H}^T\mathbf{x}_i$ and that $\mathbf{H}^T\mathbf{x}_i$ is precomputed and cached in $\mathbf{N}$. The residual function $g_{ij}(\mathbf{h}_k) = \frac{1}{2}||\mathbf{x} - \mathbf{h}_j - \mathbf{h}_k||_2^2$ is then minimized over $\mathbf{h}_k$ to find the candidate pair $(\mathbf{h}_j, \mathbf{h}_k)$ generated by each of the vectors $\mathbf{h}_j$ determined by the gradient $\nabla f_i(\mathbf{0})$. The ingredients to perform these minimizations are already in hand. This heuristic scales as $\mathcal{O}(rd)$, much better than $\mathcal{O}(d^2)$ in the original formulation. `MendelImpute` sets the default $r = 100$. In the same spirit as the first step, one can alternatively find for each j the most negative component k of the gradient $\nabla f_i(\mathbf{e}_j)$, where $\mathbf{e}_j$ is the standard unit vector with 1 in position j. This again determines a nearly optimal pair $(\mathbf{h}_j, \mathbf{h}_k)$. Under this tactic the Gram matrix $\mathbf{H}^T\mathbf{H}$ comes into play. Note that $\mathbf{H}^T\mathbf{H}\mathbf{e}_j$ reduces to its jth column $\mathbf{v}_j$. Hence, no new matrix-by-vector multiplications are necessary in calculating $\nabla f_i(\mathbf{e}_j) = -\mathbf{H}^T\mathbf{x}_i + \mathbf{v}_j = \nabla f_i(\mathbf{0}) + \mathbf{v}_j$.

Alternatively, one can find the best r unique haplotypes for a given sample $\mathbf{x}_i$ *en masse* by arranging all pairwise column distances of $\mathbf{X}$ and $\mathbf{H}$ in the matrix

$$\mathbf{R} = \begin{bmatrix} ||\mathbf{x}_1 - \mathbf{h}_1||_2^2 & \cdots & ||\mathbf{x}_n - \mathbf{h}_1||_2^2 \\ \vdots & & \vdots \\ ||\mathbf{x}_1 - \mathbf{h}_d||_2^2 & \cdots & ||\mathbf{x}_n - \mathbf{h}_d||_2^2 \end{bmatrix}_{d \times n}.$$

Then we partially sort each column of $\mathbf{R}$ to identify the top r haplotypes matching each sample $\mathbf{x}_i$. Here $\mathbf{R}$ is computed via the `Distance.jl` package of Julia, which internally performs BLAS level-3 calls analogous to computing $\mathbf{H}^T\mathbf{H}$ and $\mathbf{X}^T\mathbf{H}$. Instead of searching through all haplotypes to minimize $g_{ij}(\mathbf{h}_k)$ for a given sample $\mathbf{x}_i$, one can instead search only over the $\binom{r}{2} + r$ combinations of the top haplotypes. This allows one to entertain much larger values of r. Empirically, choosing $r = 800$ works well for most data sets.

1.6 Phasing by Dynamic Programming

We also investigated a dynamic programming strategy that gives the global solution for minimizing the number of haplotype breaks across the extended haplotypes $\mathbf{E}_1$ and $\mathbf{E}_2$. For each given haplotype pair $\mathbf{p}_1 = (\mathbf{h}_i, \mathbf{h}_j)$ in window w, we can compute the squared Hamming distance between it and the pair $\mathbf{p}_2 = (\mathbf{h}_k, \mathbf{h}_l)$ in window $w+1$; in symbols

$$d(\mathbf{p}_1, \mathbf{p}_2) = \begin{cases} 0 & \mathbf{h}_i = \mathbf{h}_k, \mathbf{h}_j = \mathbf{h}_l \quad (0 \text{ breaks}) \\ 1 & \mathbf{h}_i = \mathbf{h}_k, \mathbf{h}_j \neq \mathbf{h}_l \quad (1 \text{ break}) \\ 1 & \mathbf{h}_i \neq \mathbf{h}_k, \mathbf{h}_j = \mathbf{h}_l \quad (1 \text{ break}) \\ 4 & \mathbf{h}_i \neq \mathbf{h}_k, \mathbf{h}_j \neq \mathbf{h}_l \quad (2 \text{ breaks}). \end{cases}$$

Observe that a double break is assigned an error of 4 to favor 2 single breaks across 3 windows as opposed to a double break plus a perfect match.

Now we describe a dynamic programming strategy for finding the two paths with the minimal number

of unique haplotype breaks. We start with all candidate pairs $\mathbf{p}_i$ in the leftmost window and initialize sums $s_i = 0$ and traceback path vectors $\mathbf{t}_i$ to be empty. One then recursively visits all windows in turn from left to right. If w is the current window, then every candidate haplotype pair $\mathbf{p}_i$ in window w is connected to every candidate pair $\mathbf{p}_j$ in window $w+1$. The traceback path $\mathbf{t}_j$ is determined by the pair $\mathbf{p}_k$ minimizing $d(\mathbf{p}_i, \mathbf{p}_j)$. The traceback path $\mathbf{t}_j$ is constructed by appending $\mathbf{p}_k$ to $\mathbf{t}_k$ and setting $s_j = s_k + d(\mathbf{p}_k, \mathbf{p}_j)$. This process is continued until the rightmost window v is reached. At this point the pair $\mathbf{p}_j$ with lowest running sum s_j is declared the winner. The traceback path $\mathbf{t}_j$ allows one to construct the extended haplotypes $\mathbf{E}_1$ and $\mathbf{E}_2$ in their entirety. Unfortunately, too many haplotype pairs per window can overwhelm dynamic programming with large reference panels because one must enumerate and store all possible haplotype pairs in every genomic window for every individual. For large reference panels such as HRC, the number of possible haplotype pairs often exceeds 100,000 per window. Thus, it is impossible to store all of these pairs in memory. One partial recourse is to discard partial paths and associated partial termini $\mathbf{p}_i$ that are unpromising in the sense that their running sums s_i are excessively large. This does not completely alleviate the memory burden, and the approximate algorithm is burdened by extra bookkeeping. The bookkeeping of the exact algorithm is already demanding.

1.7 Msprime simulation script

```
# Usage: python3 msprime_script.py n ne seq recomb mut seed > full.vcf
# We used ne=10000; seq=10000000; seed=2020; recomb=2e-8; mut=2e-8
# n = 12000 or 102000 or 1002000

import msprime, sys

# parameters
sample_size = int(sys.argv[1])
effective_population_size = int(sys.argv[2])
sequence_length = int(sys.argv[3])
recombination_rate=float(sys.argv[4])
mutation_rate=float(sys.argv[5])
seed = int(sys.argv[6])

# run the simulation
ts = msprime.simulate(sample_size=sample_size, Ne=effective_population_size,
length=sequence_length, recombination_rate=recombination_rate, random_seed=seed)
model = msprime.InfiniteSites(msprime.NUCLEOTIDES)
```

Population Code	Description	Super Population
CHB	Han Chinese in Beijing, China	East Asian (EAS)
JPT	Japanese in Tokyo, Japan	East Asian (EAS)
CHS	Southern Han Chinese	East Asian (EAS)
CDX	Chinese Dai in Xishuangbanna, China	East Asian (EAS)
KHV	Kinh in Ho Chi Minh City, Vietnam	East Asian (EAS)
CEU	Utah Residents with NW European Ancestry	European (EUR)
TSI	Toscani in Italia	European (EUR)
FIN	Finnish in Finland	European (EUR)
GBR	British in England and Scotland	European (EUR)
IBS	Iberian Population in Spain	European (EUR)
YRI	Yoruba in Ibadan, Nigeria	Africans (AFR)
LWK	Luhya in Webuye, Kenya	Africans (AFR)
GWD	Gambian in Western Divisions in the Gambia	Africans (AFR)
MSL	Mende in Sierra Leone	Africans (AFR)
ESN	Esan in Nigeria	Africans (AFR)
ASW	Americans of African Ancestry in SW USA	Africans (AFR)
ACB	African Caribbeans in Barbados	Africans (AFR)
MXL	Mexican Ancestry from Los Angeles USA	Ad Mixed American (AMR)
PUR	Puerto Ricans from Puerto Rico	Ad Mixed American (AMR)
CLM	Colombians from Medellin, Colombia	Ad Mixed American (AMR)
PEL	Peruvians from Lima, Peru	Ad Mixed American (AMR)
GIH	Gujarati Indian from Houston, Texas	South Asian (SAS)
PJL	Punjabi from Lahore, Pakistan	South Asian (SAS)
BEB	Bengali from Bangladesh	South Asian (SAS)
STU	Sri Lankan Tamil from the UK	South Asian (SAS)
ITU	Indian Telugu from the UK	South Asian (SAS)

Table 1: The 26 population codes present in the 1000 genomes project.

```
ts = msprime.mutate(ts, rate=mutation_rate, model=model, random_seed=seed)

# print results (2 is for diploid, "legacy" = no matching positions)
with sys.stdout as vcffile:
    ts.write_vcf(vcffile, 2, position_transform="legacy")
```

1.8 Summary of 1000 Genomes Reference Panel

A total of 26 different populations contribute to the 1000 Genomes Project data set. These populations are further organized into five super population. While this information is freely available online, we summarize it in Table 1 for completeness.

2 Author contributions

KL, JS, ES, and HZ conceived this project. BC, KL, RW, JS, ES, and HZ devised the methods. BC, RW, and HZ developed the software. S.K added BGEN support. BC and KL accessed the data. BC wrote the original draft of the paper. BC, KL, RW, JS, ES, and HZ reviewed and edited the draft. KL previewed some of the methods incorporated in `MendelImpute` in his talk [3].

3 Acknowledgements

BC and RW were supported by NIH grant T32-HG002536. BC, ES, JS, HZ, and KL were supported by NIH grant R01-HG006139. ES, JS, HZ, and KL were supported by NIH grant R01-GM053275. JS was also supported by NIH grant R01-HG009120.

We would also like to thank Calvin Chi for his helpful discussions on ancestry estimation and Juhyun Kim for her helpful discussions on imputation quality scores.

4 Competing interests

The authors declare no competing interests.

5 Web Resources

Project name: MendelImpute.jl
Project home page: https://github.com/OpenMendel/MendelImpute.jl
Supported operating systems: Mac OS, Linux, Windows
Programming language: Julia 1.6
License: MIT
All commands needed to reproduce the following results are available at the `MendelImpute` site in the `manuscript` sub-folder. `SnpArrays.jl` is available at https://github.com/OpenMendel/SnpArrays.jl. `VCFTools.jl` is available at https://github.com/OpenMendel/VCFTools.jl. The Haplotype Reference Consortium data is available at https://www.ebi.ac.uk/ega/datasets/EGAD00001002729. Raw 1000 genomes data is available at `ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/`, and Beagle's webpage http://bochet.gcc.biostat.washington.edu/beagle/1000_Genomes_phase3_v5a/ provides a quality controlled 1000 genomes data which we used in our experiments.

References

[1] R. Finnegan and L. White. invenia/JLSO.jl: Storage container for serialized Julia objects. https://doi.org/10.5281/zenodo.3992374, 2020.

[2] A. GreenWell and M. Abbott. GroupSlices.jl: A package for the groupslices and associated functions. https://github.com/mcabbott/GroupSlices.jl, 2019.

[3] K. Lange. Lecture on Ultrafast Haplotyping. In *New Statistical Methods for Family-Based Sequencing Studies*. Banff International Research Station, http://www.birs.ca/events/2018/5-day-workshops/18w5154/videos/watch/201808091354-Lange.html, 2018.

[4] E. Y. Liu, M. Li, W. Wang, and Y. Li. MaCH-Admix: genotype imputation for admixed populations. *Genetic Epidemiology*, 37(1):25–37, 2013.