The BGEN format

A compressed binary format for typed and imputed genotype data

v1.2

This page documents version 1.2 of the BGEN format. A later version is available - see here. This version is backward-compatible with the earlier v1.1 format spec - this means every v1.1-format BGEN file is also a valid v1.2-format file. The v1.2 spec adds a new, more flexible layout for storing variant genotypes and haplotypes. (See below for a full history of versions of this spec).

Note: The UK Biobank full release imputed genotype data will be released in BGEN format, using using 8 bits per probability and zlib compression, as described in this specification. It is also planned to release phased haplotype data in BGEN format.

The UK Biobank interim release imputed genotype data was released in BGEN v1.1 format.

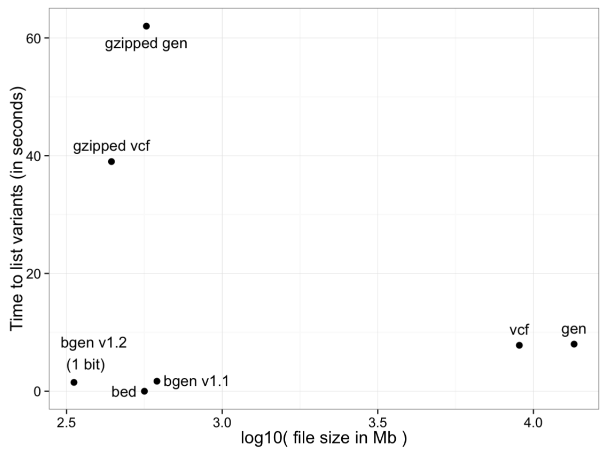
Introduction

Modern genetic association studies routinely employ data on tens to hundreds of thousands of individuals, genotyped or imputed at tens of millions of markers genome-wide. Traditional data formats based on text representation of these data - such as the GEN format output by IMPUTE, or the Variant Call Format - are sometimes not well suited to these data quantities. Indeed, for simple programs the time spent parsing these formats can dominate program execution time.

This page describes a binary GEN file format (the "BGEN" format) which aims to address these problems. BGEN is a robust format that has been designed to have a specific blend of features that we believe make it useful for this type of study. It is targetted for use with large, potentially imputed genetic datasets. Key features include:

- The ability store both directly typed and imputed data.
- The ability to store both unphased genotypes and phased haplotype data.
- Small file sizes through the use of efficient, variable-precision packed bit representations and compression.
- The use of per-variant compression makes the format simple to index and easy to catalogue.

For example, the following plot shows the time taken to list variant identifying data - i.e. the genomic position, ID fields and alleles - for various common formats (Y-axis), against file size (X axis), for a dataset of 18,496 samples typed at 121,668 SNPs on chromosome 1. Both variants of BGEN defined below are shown.



For PLINK binary (.bed) files, identifying data is stored in a separate file (the .bim file) so the time is effectively zero. For text-based formats there is a significant trade-off between the use of file compression and read performance. BGEN stores the entire dataset of 2,250 million genotypes in 334Mb, slightly over one bit per genotype, and in this test took 1.5s.

(Performance optimisation of all formats may of course be possible, so the above plot will not represent the best possible timings, but should be regarded as illustrative.)

The BGEN format has been used in several major projects, including the Wellcome Trust Case-Control Consortium 2 and the MalariaGEN project. It has been adopted as the release format for genome-wide imputed genotypes for the UK Biobank.

Acknowledgements. The following people contributed to the design and implementation of the

BGEN format:

- Gavin Band
- Jonathan Marchini

Software support

A freely available C++ implementation of the BGEN format is available on bitbucket. This repository also contains utilities:

- A bgen indexing tool bgenix.
- A tool cat-bgen for efficiently concatenating bgen files. We have found this useful for joining together chunks of imputed data after a genome-wide imputation run.
- An example program, bgen_to_vcf, which converts bgen-formatted files to Variant Call Format. This is intended as an example of use of the API.

In addition, bgen support has been implemented in several other software packages:

- QCTOOL can be used to read and write BGEN files and to convert between BGEN and other formats.
- SNPTEST has supported BGEN v1.1 since version 2.4.0, and BGEN v1.2 since version 2.5.1.
- PLINK v1.9 now includes support for BGEN.
- Mega2 now includes support for BGEN.

(Please contact me if your software supports BGEN and you'd like it added to this list.)

Change history

A history of revisions of the v1.2 format spec is as follows:

BGEN v1.2 (November 2016):
Major update extending the BGEN format to add:
- Support for variable ploidy and explicit missing data.
- Support for multi-allelic variants (e.g. complex structural variants).
- Allow for control over file size by supporting genotype probabilities stored at configurable precision.
- Support for storing sample identifiers.

A draft version of this spec was published beginning May 2015. The following changes have been made since the earlier draft:
- **2015-11-05** (v1.2 beta1): modified the treatment of missing data in Layout 2 (v1.2-style) variant data blocks.
- **2016-03-21** (v1.2 beta2): modified the order of stored probabilities for samples with ploidy greater than 2; clarified specification of the phased flag for samples with ploidy less than 2.

BGEN v1.1 (March 2012):
The first widely used version of the BGEN format. The UK Biobank interim imputed data was released in this format. Relative to v1.0, this version is designed to cope with the long alleles present at indels and structural variants in recent releases of the 1000 genomes project. Features of this version are:
- Support for biallelic SNPs and indels with alleles of arbitrary length (up to 2^{32}-1).
- Store probabilities to at least 4 decimal places worth' of accuracy

BGEN v1.0 (2009):
The original BGEN format. This version is now deprecated and will be removed from a future version of this spec; there probably aren't any files in the wild in this format.

Detailed specification

Overview

A BGEN file consists of a header block, giving general infomation about the file, and an optional sample identifier block. These are followed by a series of *variant data* blocks, stored consecutively in the file, which each contain data for a single genetic variant. To allow for potential future additions to the spec, the first variant data block is located using an offset stored in the first four bytes of the file.

The format in which variant data blocks are stored is determined by a set of flag bits stored in the header block. Currently two formats are supported - Layout 1 blocks which are a direct translation to binary of the GEN format; and Layout 2 blocks, which are both more space-efficient and more flexible, including support for genotype and haplotype data, multi-allelic variants, and non-diploid samples. An older format, used in the v1.0 spec, is now deprecated and is no longer documented in this spec.

Data types

All numbers in a BGEN file are stored as unsigned integers in little endian (least significant byte first) order. This choice coincides with the memory layout used on most common architectures - see the wikipedia page for more details.

Variant identifiers, chromosome identifiers, and other string fields are stored as a two- or four-byte integer length followed by the data itself (which does not include a C-style trailing zero byte).

Genotype probabilities are stored in an efficient packed bit representation described in detail below.

Finally, some fields in BGEN are interpreted as flags encoded as a bitmask.

The first four bytes

The first four bytes of the file encode an unsigned integer indicating the offset, relative to the 5th byte of the file, of the start of the first variant data block, or the end of the file if there are 0 variant data blocks. For example, if this offset is 20 (the minimum possible because the header block always has size at least 20) then the variant data blocks start at byte 25.

No. of bytes	Description
4	An unsigned integer *offset* indicating the offset, relative to the fifth byte of the file, of the first byte of the first variant data block (or the end of the file if there are no variant data blocks).
4	**TOTAL**

The header block

The header block contains global information about the file, including the number of samples and the number of variant data blocks the file contains, and flags indicating how data is stored.

No. of bytes	Description
4	An unsigned integer L_H indicating the length, in bytes, of the header block. This must not be larger than *offset*.
4	An unsigned integer *M* indicating the number of variant data blocks stored in the file.
4	An unsigned integer *N* indicating the number of samples represented in the variant data blocks in the file.
4	'Magic number' bytes. This field should contain the four bytes 'b', 'g', 'e', 'n'. For backwards compatibility, readers should also accept the value 0 (four zero bytes) here.
L_H-20	Free data area. This could be used to store, for example, identifying information about the file
4	A set of *flags*, with bits numbered as for an unsigned integer. See below for flag definitions.
$\boldsymbol{L_H}$	**TOTAL**

Header block -- flag definitions

The following flags can be contained in the *flags* field in the header block. **Note**: bits and field values not specified here are reserved for possible future use; they should be set to zero.

Bit	Name	Value	Description
0-1	*CompressedSNPBlocks*	0	Indicates SNP block probability data is not compressed.
		1	Indicates SNP block probability data is compressed using zlib's compress() function.
2-5	*Layout* (previously called *LongIds*)	0	Indicates SNP blocks are layed out according to **Layout 0**, first used in the v1.0 spec. This allows only single-character alleles. **Use of this format is deprecated, in the sense that it should not be**

			used for new files. We will remove this from a future version of the spec.
		1	Indicates SNP blocks are layed out according to **Layout 1**, i.e. as in the v1.1 spec. This allows for multiple characters in alleles and is supported in SNPTEST from version 2.3.0, and in QCTOOL from version 1.1.
		2	Indicates SNP blocks are layed out according to **Layout 2**, introduced in version 1.2 of the spec (i.e. in this document). This format supports multiple alleles, phased and unphased genotypes, explicit specification of ploidy and missing data, and configurable levels of compression. It is recommended that all new files are stored with *Layout*=2.
			Values > 2 are reserved for future use.
31	*SampleIdentifiers*	0	Indicates sample identifiers are not stored in this file.
		1	Indicates a sample identifier block follows the header. It is recommended that all new files are created with *SampleIdentifiers*=1.

Sample identifier block

If *SampleIdentifiers*=1 in the flags field, the header block is immediately followed by a sample identifier block. This stores a single identifier per sample.

Note: BGEN treats sample identifiers as a string of bytes, and does not impose any additional restrictions. However, for the simplest interoperability with other software (e.g. for R's make.names) it is often sensible to restrict to ASCII alphanumeric characters, underscores, and full stop.

No. of bytes	Description
4	An unsigned integer L_{SI} indicating the length in bytes of the sample identifier block. This must satisfy the constraint $L_{SI}+L_H \leq$ *offset*.
4	An unsigned integer N indicating the number of samples represented in the file. This must be the same as the number N in the header block.
2	An unsigned integer indicating the length L_{s1} of the identifier of sample 1.
L_{s1}	Identifier of sample 1.
2	An unsigned integer indicating the length L_{s2} of the identifier of sample 2.
L_{s1}	Identifier of sample 2.
...	
2	An unsigned integer indicating the length L_{sN} of the identifier of sample N.
L_{sN}	Identifier of sample N.
$L_{SI} = 8 + 2 \times N + \sum_n L_{sn}$	**TOTAL**

Variant data blocks

Following the header comes a sequence of *M* variant data blocks (where *M* is the number specified in the header block). This document describes SNP blocks for spec versions 1.1 and above. Version 1.0 is deprecated and should not be used in new files.

Variant data blocks are comprised of: a section of identifying data (containing variant IDs, position, and alleles), followed by a section containing the genotype probability data itself. Most files will have *CompressedSNPBlocks=1*, indicating that genotype probability data is stored compressed. (The variant identifying data is never compressed, however.)

Variant identifying data

No. of bytes	Description
4	The number of individuals the row represents, hereafter denoted *N*. This is only present if *Layout=1* (otherwise it appears instead in the genotype probability block below).
2	The length L_{id} of the variant identifier. (The variant identifier is intended to store e.g. chip manufacturer IDs for assayed SNPs).
L_{id}	The variant identifier.
2	The length L_{rsid} of the rsid.
L_{rsid}	The rsid.
2	The length L_{chr} of the chromosome
L_{chr}	The chromosome
4	The variant position, encoded as an unsigned 32-bit integer.
2	The number *K* of alleles, encoded as an unsigned 16-bit integer. If *Layout*=1, this field is omitted, and assumed to equal *2*.
4	The length L_{a1} of the first allele.
L_{a1}	The first allele.
4	The length L_{a2} of the second allele.
L_{a2}	The second allele.
...	...(possibly more alleles)...
4	The length L_{aK} of the *K*th allele.
L_{aK}	The *K*th allele.
$16 + 4K + L_{id} + L_{rsid} + L_{chr} + \sum_k L_{ak} + D$	**TOTAL**

Genotype data block (Layout 1)

Layout 1 blocks are used when *Layout=1*. Only two alleles (*K=2*) are supported. All samples are stored as if diploid; haploid samples should be stored as if having homozygous genotype. Missing samples are encoded as three zero probabilities. This is a direct translation to binary format of a GEN file.

No. of bytes	Description
4	The total length *C* of the compressed genotype probability data for this variant. Seeking forward this many bytes takes you to the next variant data block. If *CompressedSNPBlocks=0* this field is omitted and the length of the uncompressed data is *C=6N*.
C* or *C+4	**TOTAL**
C	Genotype probability data for the SNP for each of the *N* individuals in the cohort in the format described below. If *CompressedSNPBlocks=0* this consists of *C=6N* bytes in the format described below. Otherwise this is *C* bytes which can be uncompressed using zlib to form *6N* bytes stored in the format

No. of bytes	Description
	described below. (Zstandard compression, encoded by the value *CompressedSNPBlocks = 2*, is not supported for v1.1 style blocks.)
C or *C+4*	**TOTAL**

Probability data storage

For Layout 1 blocks, probability data is stored as a sequence of 2-byte unsigned integers. These should be interpreted in triples, the first member being the probability of a homozygous 'AA' allele, the second the probability of 'AB', the third the probability of 'BB', where A and B are the two alleles at the variant. When *CompressedSNPBlocks* is not set, these 6 * *N* bytes are stored in the file directly. When *CompressedSNPBlocks>0*, these 6**N* bytes are first compressed using zlib and the length of the compressed data is stored as the 4-byte integer *C*, followed by the compressed data itself.

To convert the stored 2-byte integers into probabilities, the following calculation should be performed:

1. Convert the number into a floating-point format (e.g. float or double).
2. Divide by 32,768.

Note that the range of a two-byte unsigned integer is 0 - 65,535 inclusive. Thus the resulting probabilities can take on values between 0 and 65,535/32768 ~ 1.9999 inclusive and they are accurate to four decimal places.

To convert a floating point probability to its integer representation, do the following:

1. Multiply by 32,768.
2. Check that the number is in the half-open interval [0,65535.5) and round to the nearest integer.

All numbers are stored in little-endian (least significant byte first) order. Probabilities for samples with missing genotype data should be stored as zero.

Genotype data block (Layout 2)

Layout 2 blocks are used when *Layout=2*. This format supports arbitrary numbers of alleles (up to 65535), samples of arbitrary ploidy (up to 63), and both phased and unphased data.

No. of bytes	Description
4	The total length *C* of the rest of the data for this variant. Seeking forward this many bytes takes you to the next variant data block.
4	The total length *D* of the probability data after uncompression. If *CompressedSNPBlocks = 0*, this field is omitted and the total length of the probability data is *D=C*.
C or *C-4*	Genotype probability data for the SNP for each of the *N* individuals in the cohort. If *CompressedSNPBlocks = 0*, this is *D* bytes stored in the format described below. If *CompressedSNPBlocks = 1*, this is *C-4* bytes which can be uncompressed using zlib to form *D* bytes in the format described below.

Probability data storage

Layout 2 probability data storage is structured as described below. If *CompressedSNPBlocks = 0* the structure is stored directly, and *C* reflects the length of this structure. If *CompressedSNPBlocks > 0* the whole structure is stored after compression. In this case *D* reflects the length of the uncompressed structure and the length of the compressed structure is *C-4*.

No. of bytes	Description
4	The number of individuals for which probability data is stored. This must equal *N* as defined in the header block.
2	The number of alleles, encoded as an unsigned 16-bit integer. This must equal *K* as defined in the variant identifying data block.
1	The minimum ploidy P_{min} of samples in the row. Values
$D=10+N+\sum_i P_i$	**TOTAL**

between 0 and 63 are allowed.

No. of bytes	Description
1	The maximum ploidy P_{max} of samples in the row. Values between 0 and 63 are allowed.
N	A list of *N* bytes, where the *n*th byte is an unsigned integer representing the ploidy and missingness of the *n*th sample. Ploidy (possible values 0-63) is encoded in the least significant 6 bits of this value. Missingness is encoded by the most significant bit; thus a value of 1 for the most significant bit indicates that no probability data is stored for this sample. (**Note**: there is no way to indicate that the ploidy itself is missing.)
1	Flag, denoted *Phased* indicating what is stored in the row. If *Phased*=1 the row stores one probability per allele (other than the last allele) per haplotype (e.g. to represent phased data). If *Phased*=0 the row stores one probability per possible genotype (other than the 'last' genotype where all alleles are the last allele), to represent unphased data. Any other value for *Phased* is an error.
1	Unsigned integer *B* representing the number of bits used to store each probability in this row. This must be between 1 and 32 inclusive.
X	Probabilities for each possible haplotype (if *Phased*=1) or genotype (if *Phased*=0) for the samples. Each probability is stored in *B* bits. Values are interpreted by linear interpolation between 0 and 1, i.e. value *b* corresponds to probability *b* / (2^B-1). When storing the value, probabilities should be rounded according to the algorithm described below. Probabilities are stored consecutively for samples 1, 2, ..., *N*. For each sample the order of stored probabilities is described below. Probabilities for samples with missing data (as defined by the missingness/ploidy byte) are written as zeroes (note this represents a change from the earlier draft of this spec; see the rationale below).
$D=10+N+\sum_i P_i$	**TOTAL**

Per-sample order of stored probabilities

Consider a sample with ploidy *Z* and a variant with *K* alleles.

- For **phased data**, probabilities are stored in the order of haplotypes and then alleles, ie:
  $P_{11}, P_{12}, ..., P_{1(K-1)}, P_{21}, ..., P_{2(K-1)}, ..., P_{Z1}, ..., P_{Z(K-1)}$.
  where P_{ij} is the probability that haplotype *i* has allele *j*. For each haplotype *i* the probability of the Kth allele (P_{iK}) is not stored; instead it is inferred as one minus the sum of other probabilities for that haplotype. Thus a total of *Z(K-1)* probabilities are stored.

- For **unphased data**, enumerate the possible genotypes as the set of *K*-vectors of nonnegative integers $(x_1, x_2, ..., x_K)$, where x_i represents the count of the *i*-th allele in the genotype. Probabilities are stored in colex order of these vectors. The last probability (corresponding the the *K*-th allele homozygotes) is not stored; instead it is inferred as one minus the sum of other probabilities. Thus a total of *(Z+K-1)* choose *(K-1)*-1 probabilities is stored.

  Example. For example if *Z=3* and *K=3* then the enumerated genotypes with allele count representations are:

Index	Genotype	Allele counts
0	111	(3,0,0)
1	112	(2,1,0)
2	122	(1,2,0)
3	222	(0,3,0)
4	113	(2,0,1)
5	123	(1,1,1)
6	223	(0,2,1)
7	133	(1,0,2)
8	233	(0,1,2)

9	333	(0,0,3)

The stored probabilities are thus

$$P_{111},P_{112}, P_{122}, P_{222}, P_{113}, P_{123}, P_{223}, P_{133}, P_{233}$$

with P_{333} inferred as one minus the sum of the other probabilities.

The colex order has the important property that the genotypes that for each i the genotypes carrying the i-th allele appear later in the order than those that carry only alleles $1,\ldots,i-1$. See the rationale below for a further discussion of this choice of storage order.

Representation of probabilities

For both genotype and haplotype data, each probability value is stored using B bits as follows. An integer of length B bits can represent the values $0, \ldots, 2^B-1$ inclusive. To interpret a stored value x as a probability:

1. Convert x to an integer in floating-point representation.
2. Divide by 2^B-1.

Thus, probabilities stored in Layout 2 blocks take possible values of the form $x/(2^B-1) \in [0,1]$.

Storing probabilities to the limited precision afforded by B bits requires a rounding rule, which we specify as follows. Given a vector $v=(v_1, \ldots v_d)$ of d probabilities that sum to one, we round by finding the closest point to v of the form $x/(2^B-1)$ where the entries of x are nonnegative integers summing to (2^B-1). The integer vector x can be found by the following algorithm:

1. Multiply v by 2^B-1.
2. Compute the total fractional part $F = \sum_i (v_i - \text{floor}(v_i))$.
3. Form x by rounding the F entries of v with the largest fractional parts up to the nearest integer, and the other $d-F$ entries down to the nearest smaller integer.

The results of Bomze et al, 2014 imply that $x/(2^B-1)$ is the nearest point to v that can be stored in the BGEN format with B bits.

The maximum error in a probability stored using this rounding rule is $1/(2^B-1)$.

In practice we there may be some rounding error in probabilities input into the BGEN format. We therefore renormalise input probabilities to sum to one.

Rationale and FAQ

Q. Should I use BGEN v1.1 or v1.2? Or something else?

A. The short answer is that we believe that for most purposes BGEN v1.2 is a good choice for storage of hard-called or imputed genotypes.

For a more detailed answer, the following table tabulates features of various different formats:

	PLINK binary	GEN	BGEN v1.1	BGEN v1.2	VCF	BCF
Supports unphased genotype calls	✓	✓*	✓*	✓	✓	✓
Supports unphased genotype probabilities		✓	✓	✓	✓	✓
Supports NULL/outlier probability e.g. NULL class from CHIAMO / GenoSNP		✓	✓		✓	✓
Supports non-diploid samples		†	†	✓	✓‡	✓‡
Supports phased data?				✓	✓‡	✓‡
Supports multi-allelic				✓	✓	✓

variants						
Efficient representation?	✓		✓	✓		✓

[*]Hard-called genotypes are converted to probabilities in GEN and BGEN v1.1. [†]By convention, males on the X chromosome are stored as homozygote females in GEN and BGEN v1.1. [‡]At the time of writing, the storage of genotype likelihoods and probabilities for non-diploid samples and/or phased data in VCF/BCF is not fully specified.

Thus BGEN v1.2 is appropriate except for storing genotype probabilities from cluster-based calles such as CHIAMO or GenoSNP, which can assign nonzero probability to a NULL or outlier class. Use BGEN v1.1 or another format for these data.

Another consideration over what to use is tool support. Support for BGEN v1.1 has been available for some time in QCTOOL, SNPTEST and other packages including PLINK. Support for BGEN v1.2 is becoming available across a range of tools mid-2016. We expect tool support for BGEN to continue to increase in future, driven in part by the use of BGEN for the UK Biobank data releases and the availability of a reference implementation.

Q. Why compress the data for each variant seperately? Doesn't this end up using more space?

A. Compressing each variant seperately has the advantage that the variant identifying data fields - which are stored uncompressed in BGEN - can be efficiently accessed without decompressing the whole file. It's true that theoretically higher rates of compression could be acheived by compressing as much data as possible. However, for large cohorts, each variant has plenty of data to compress, so the difference is expected to be small. (Alternative indexing approaches - like bgzip, which compresses data in blocks - also incur a size overhead.)

Q. Why store the number of individuals on each row? Why store the number of alleles twice? Doesn't this waste space?

A. Yes, each repeated field wastes 4 bytes per variant. However, this is expected to be a tiny fraction of overall file size. The inclusion of repeated fields makes parsing simpler (since parsers don't need to be stateful) and also provides a useful way of testing that file structure is correct.

Q. (For Layout 2 variant data blocks) Why the choice of colex order of genotypes?

A. Several considerations motivated this choice - including simplicity of specification, ease of implementation, and compatibility with established standards.

The colex order coincides with the order specified for GL and GP fields for diploid samples in the VCF. (At the time of writing the VCF specification of this field only applies to diploid samples). This makes it easy to convert between BGEN and VCF formats.

The chosen order has a useful 'nesting' property: adding an allele does not change the order of probabilities for genotypes not carrying the new allele. More precisely, at a variant with $K+1$ alleles, genotypes carrying only alleles among the first K appear earlier in the ordering than those carrying the Kth allele - and with the same order as if the 'extra' allele wasn't present. A practical consequence of this is that mapping of each genotype to the index of the corresponding probability can be accomplished using a single lookup table that is independent of K (though it does depend on the ploidy). We used such a lookup table in the QCTOOL implementation to convert hard-called genotypes to probabilities.

Q. (For Layout 2 variant data blocks) Why omit the last probability instead of the first?

A. The intention is to make parsing the file as simple and fast as possible. If we omitted the first probability, parsers would have to read and store all the other probabilities before computing and emitting the first probability. With the scheme described here, parsers can simply emit the probabilities as they are read from the file, before finally emitting the last probability as one minus the sum of the stored probabilities.

Q. (For Layout 2 variant data blocks) Why store dummy zero data for samples with missing genotypes? Doesn't this increase file size unneccessarily?

A. (Note this behaviour was changed in the beta version of the spec relative to the earlier draft.) The intention is to make parsing the file as simple and fast as possible. One option for loading genotypes from a BGEN file is simply to load the uncompressed genotype data into memory. Potentially, encoded genotypes could then be used directly, e.g. as keys in a lookup table. Storing placeholder data for missing samples makes this simpler to implement by maintaining a predictable index for the data for each sample within the encoded representation. (We note this will be more complicated if ploidy varies across samples).