bedp: A Binary Phased Haplotype Table Format Supporting Efficient Random Access

Ryan Lewis
08/2020

Purpose

- **Create a file format that is capable of holding Phased Haplotype data and is computational efficient at random accessing data**

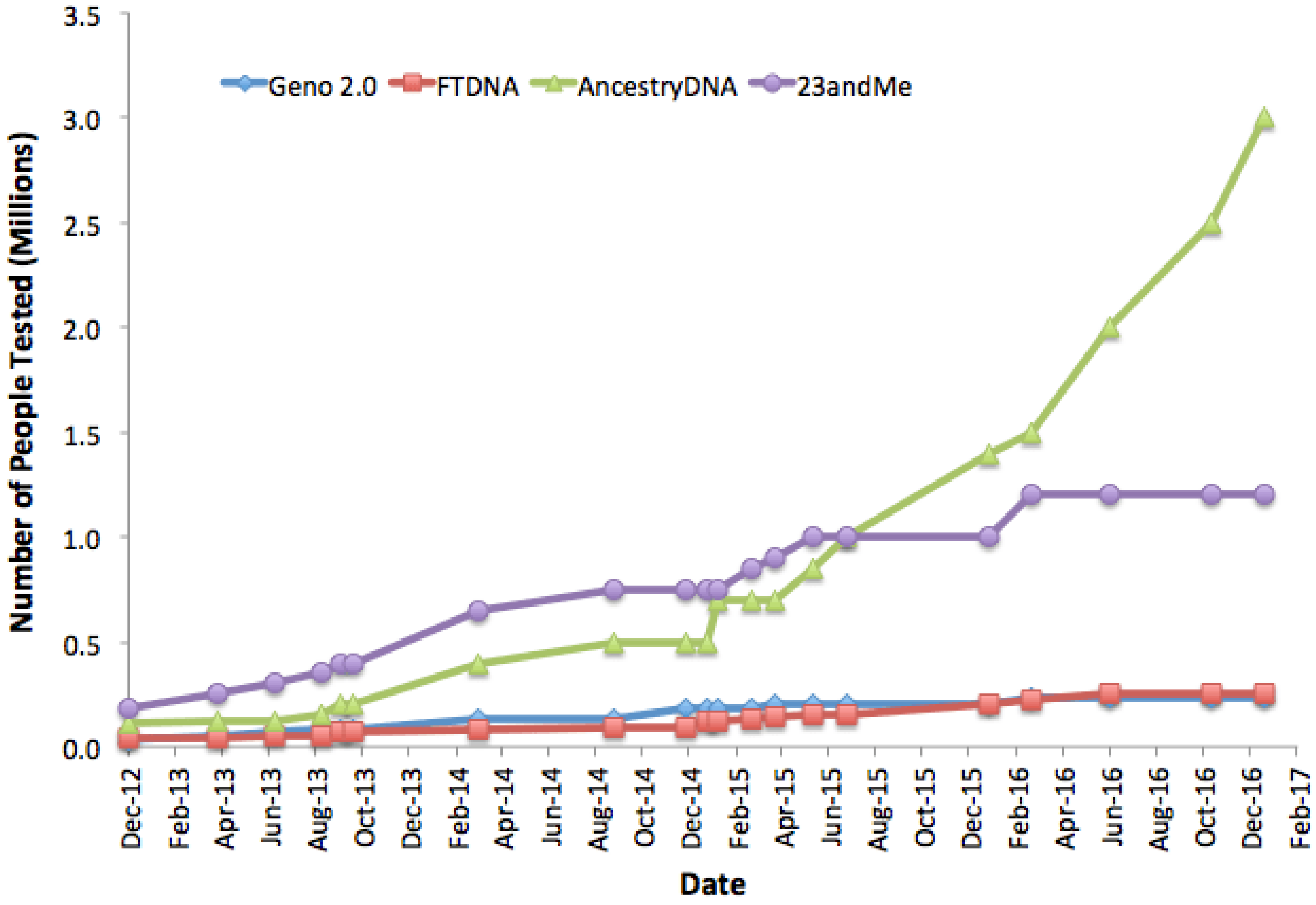
Why is this necessary?

- Large biobanks contain data used in genetic research and the value of this data is in the vast number of individuals who volunteered their genetic data for research purposes.

- UK BioBank (~500,000) https://www.ukbiobank.ac.uk/
- TopMed (~150,000) https://www.joinallofus.org/
- FINNGEN (~225,000) https://www.finngen.fi/en
- Canada CPTP (~180,000) https://canpath.ca/
- EuroBioBank (~150,000) http://www.eurobiobank.org/

Why is this necessary? (cont.)

- Not only are biobanks an important factor in creating this file format, but the growing number of consumer genetic testing kits being sold is a factor as well



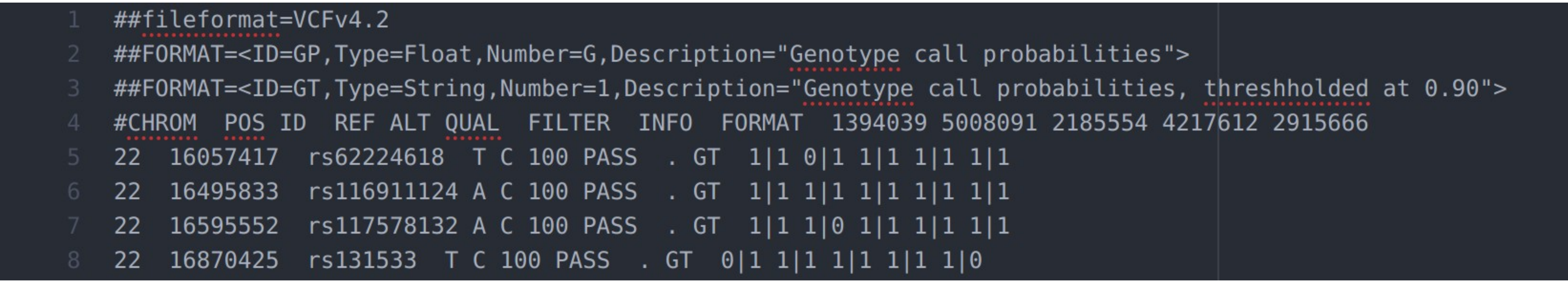
© 2016-2017 by Leah Larkin; Source: ISOGG wiki "Autosomal DNA testing comparison chart" edit history

Why is this necessary?(cont.)

- As more and more data is added to these large cohorts it will become computationally more difficult to analyze, especially when identifying identity-by-descent segments among the cohort.

Other File Formats

- **VCF**
  - Holds phased haplotype data but is not computationally efficient at random access



```
##fileformat=VCFv4.2
##FORMAT=<ID=GP,Type=Float,Number=G,Description="Genotype call probabilities">
##FORMAT=<ID=GT,Type=String,Number=1,Description="Genotype call probabilities, threshholded at 0.90">
#CHROM  POS ID  REF ALT QUAL  FILTER  INFO  FORMAT  1394039 5008091 2185554 4217612 2915666
22  16057417  rs62224618  T C 100 PASS  . GT  1|1 0|1 1|1 1|1 1|1
22  16495833  rs116911124 A C 100 PASS  . GT  1|1 1|1 1|1 1|1 1|1
22  16595552  rs117578132 A C 100 PASS  . GT  1|1 1|0 1|1 1|1 1|1
22  16870425  rs131533  T C 100 PASS  . GT  0|1 1|1 1|1 1|1 1|0
```

- **BED**
  - Does not hold phased haplotype data but is computationally efficient at random access

Why not use bed?

- **The PLINK .bed format stores data in 2 bits, giving 4 options**
  - 00 homozygous for first allele
  - 01 missing genotype
  - 10 heterozygous
  - 11 homozygous for second allele
- **The 4 options provided do not allow for storing Phased Haplotype Data**

What is bedp?

- **bedp is a file format whose design is based on the framework of the PLINK file format, .bed**
  - This allows bedp to utilize the PLINK tool set
- **It is designed to hold genetic data in a binary format and allows users to access said data in a computationally efficient manner**

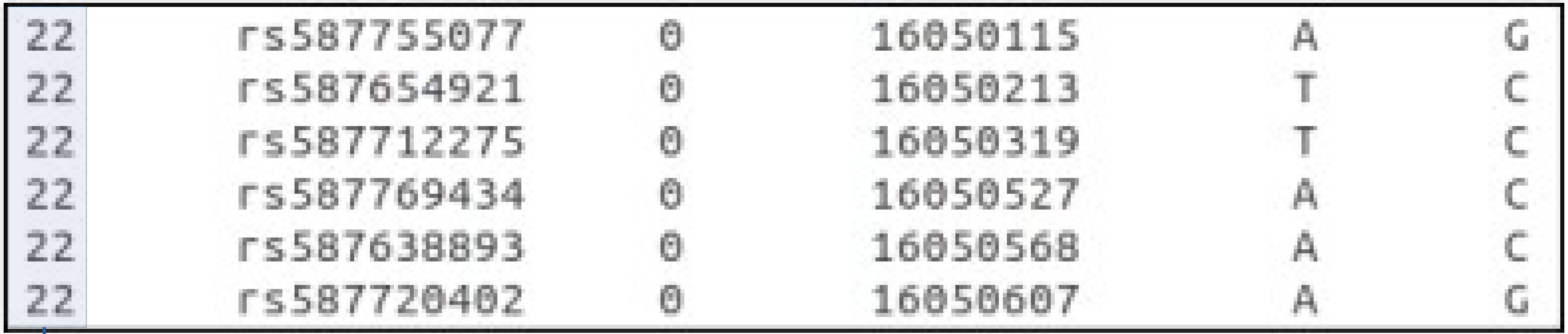
Reference: https://www.cog-genomics.org/plink2/formats

bedp Structure

- **bedp is comprised of 3 files**
  - .bim
  - .fam
  - .bed - *must use extension name ‘.bed’, not ‘.bedp’, in order to use the PLINK tool set*
- **The file name for the 3 files must be identical in order to identify correspondence**
  - ex. file1.bed, file1.bim, file1.fam

.bim File

- **The .bim holds data on the SNPs present in the data set. Each line in the file represents 1 SNP and each line has 6 columns (tab delimiter)**

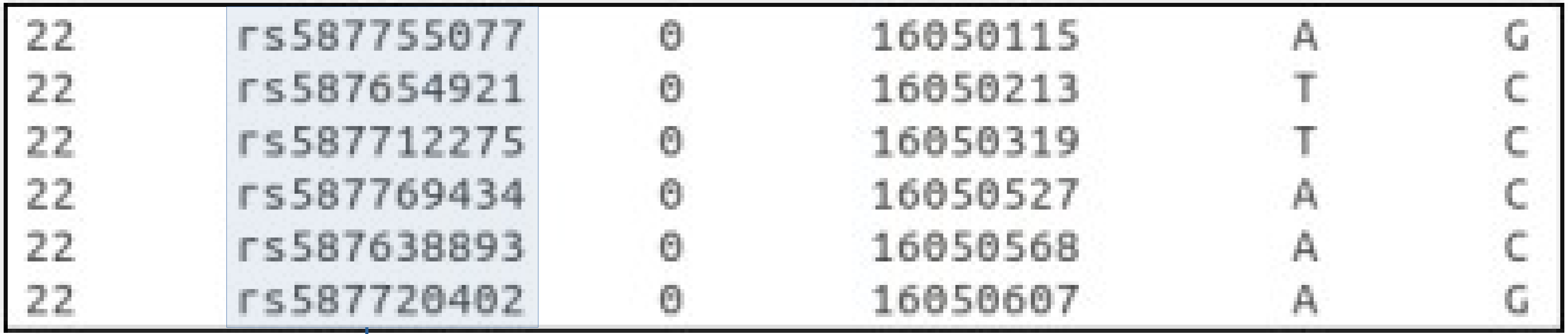


```
22	rs587755077	0	16050115	A	G
22	rs587654921	0	16050213	T	C
22	rs587712275	0	16050319	T	C
22	rs587769434	0	16050527	A	C
22	rs587638893	0	16050568	A	C
22	rs587720402	0	16050607	A	G
```

Chromosome Identification

.bim File

- The .bim holds data on the SNPs present in the data set. Each line in the file represents 1 SNP and each line has 6 columns (tab delimiter)

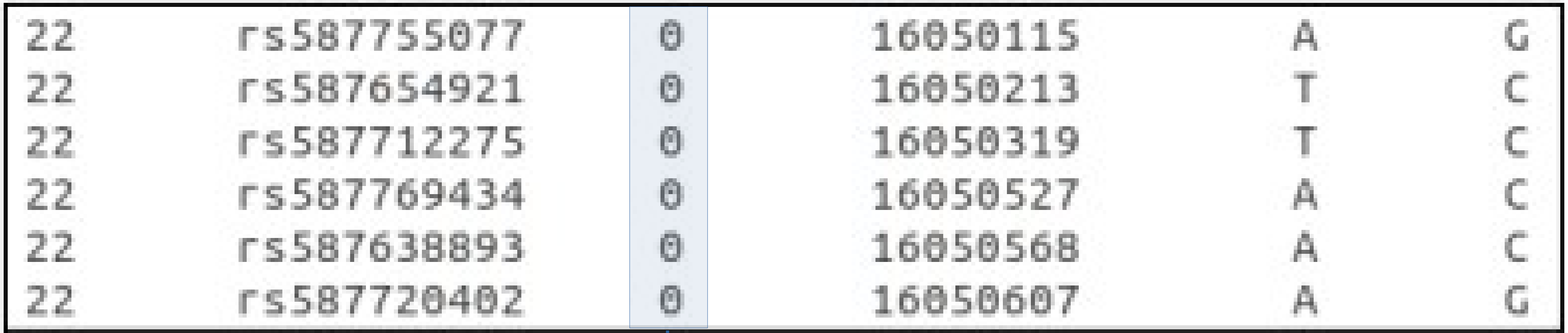


```
22      rs587755077     0       16050115        A       G
22      rs587654921     0       16050213        T       C
22      rs587712275     0       16050319        T       C
22      rs587769434     0       16050527        A       C
22      rs587638893     0       16050568        A       C
22      rs587720402     0       16050607        A       G
```

RSID or SNP Identifier

.bim File

- **The .bim holds data on the SNPs present in the data set. Each line in the file represents 1 SNP and each line has 6 columns (tab delimiter)**

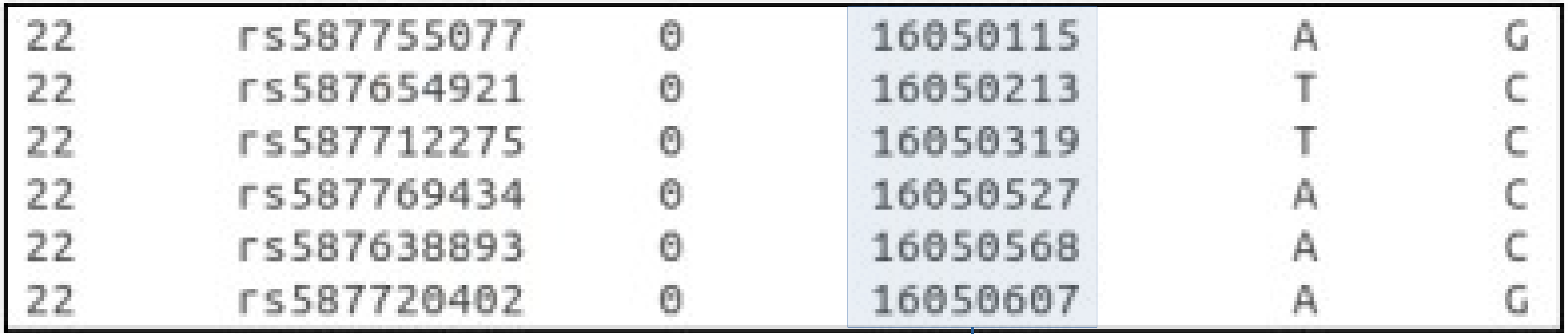


```
22	rs587755077	0	16050115	A	G
22	rs587654921	0	16050213	T	C
22	rs587712275	0	16050319	T	C
22	rs587769434	0	16050527	A	C
22	rs587638893	0	16050568	A	C
22	rs587720402	0	16050607	A	G
```

Position in cM, default = 0

.bim File

- **The .bim holds data on the SNPs present in the data set. Each line in the file represents 1 SNP and each line has 6 columns (tab delimiter)**

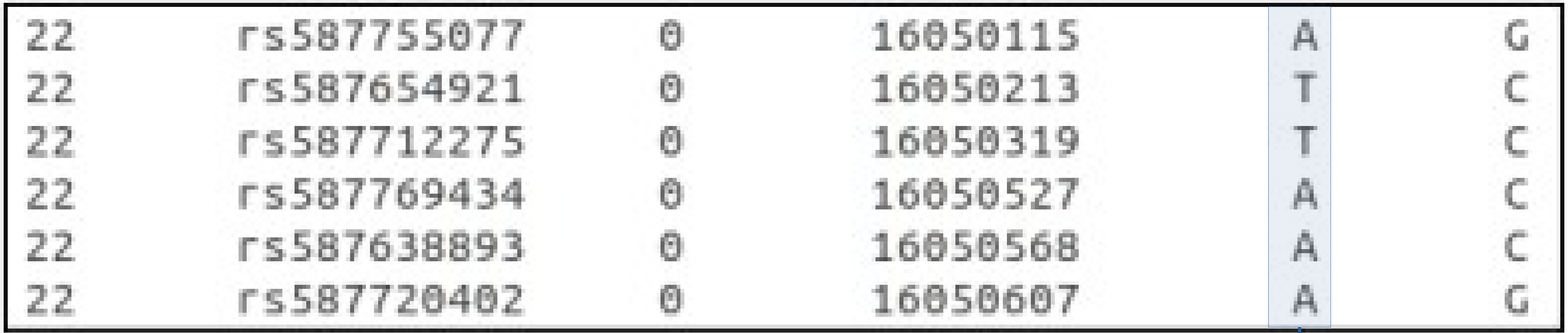


```
22	rs587755077	0	16050115	A	G
22	rs587654921	0	16050213	T	C
22	rs587712275	0	16050319	T	C
22	rs587769434	0	16050527	A	C
22	rs587638893	0	16050568	A	C
22	rs587720402	0	16050607	A	G
```

Base-pair coordinate (position)

.bim File

- **The .bim holds data on the SNPs present in the data set. Each line in the file represents 1 SNP and each line has 6 columns (tab delimiter)**

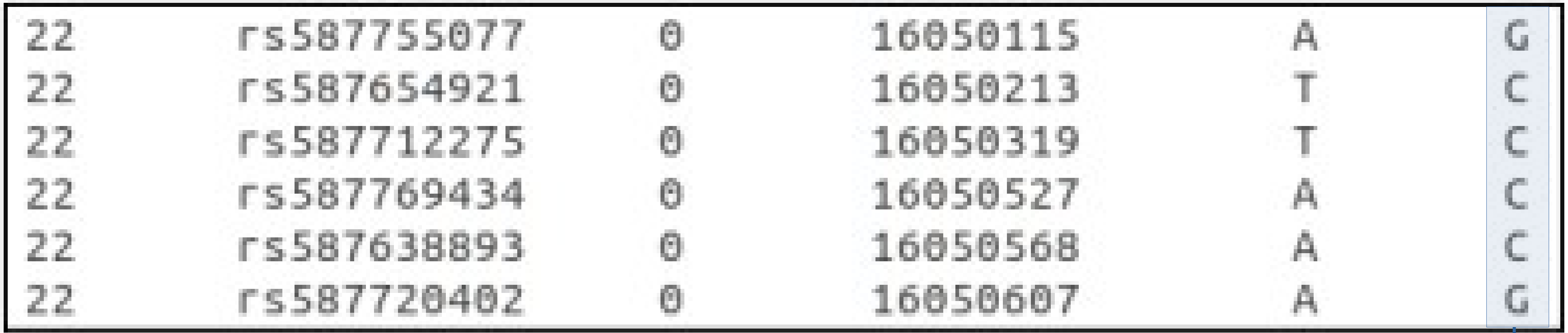


```
22    rs587755077    0    16050115    A    G
22    rs587654921    0    16050213    T    C
22    rs587712275    0    16050319    T    C
22    rs587769434    0    16050527    A    C
22    rs587638893    0    16050568    A    C
22    rs587720402    0    16050607    A    G
```

1st Allele: corresponds to the clear bits, 0 (minor)

.bim File

- **The .bim holds data on the SNPs present in the data set. Each line in the file represents 1 SNP and each line has 6 columns (tab delimiter)**

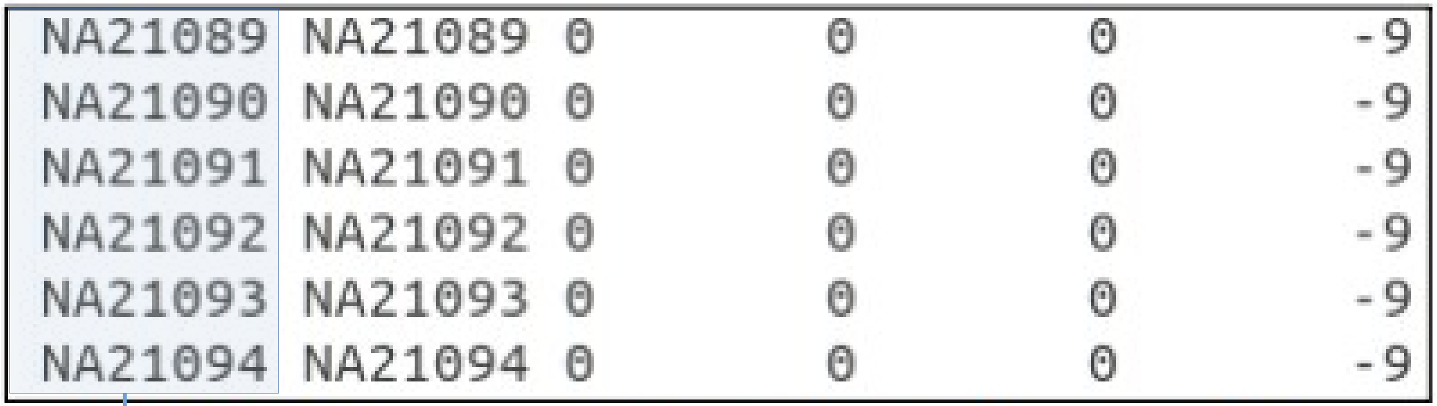


```
22	rs587755077	0	16050115	A	G
22	rs587654921	0	16050213	T	C
22	rs587712275	0	16050319	T	C
22	rs587769434	0	16050527	A	C
22	rs587638893	0	16050568	A	C
22	rs587720402	0	16050607	A	G
```

2nd Allele: corresponds to the set bits, 1 (major)

.fam File

- **The .fam file holds data on the individuals contained in the data set. Each line represents 1 individual and has six columns(tab delimiter)**

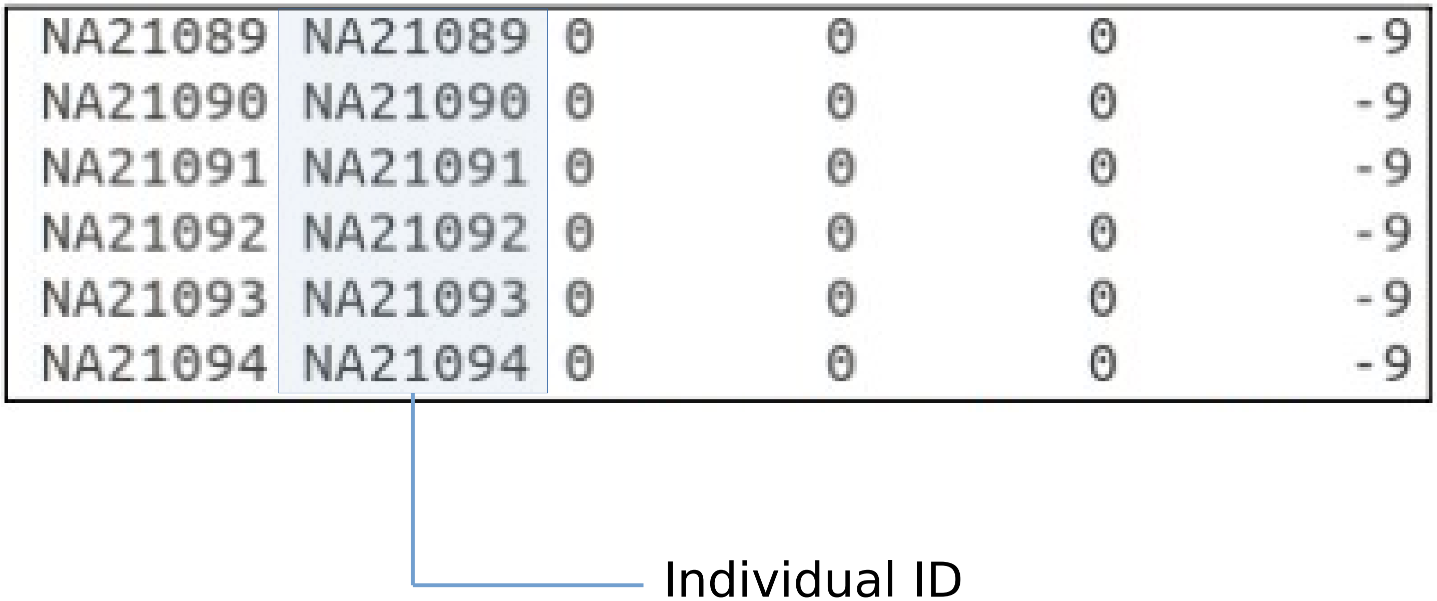


```
NA21089 NA21089 0       0       0       -9
NA21090 NA21090 0       0       0       -9
NA21091 NA21091 0       0       0       -9
NA21092 NA21092 0       0       0       -9
NA21093 NA21093 0       0       0       -9
NA21094 NA21094 0       0       0       -9
```

Family ID, default = Individual ID

.fam File

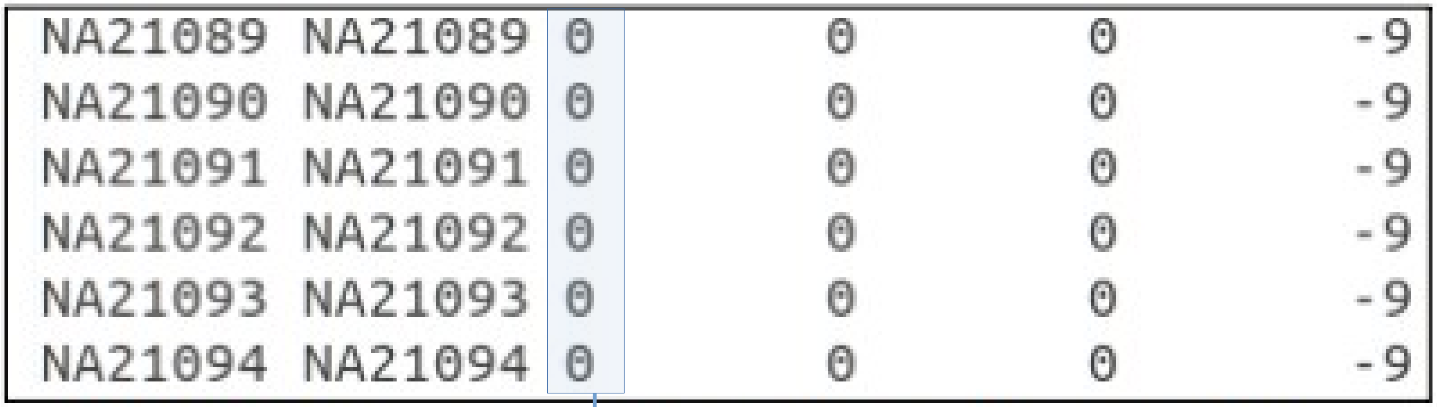
- The .fam file holds data on the individuals contained in the data set. Each line represents 1 individual and has six columns(tab delimiter)



```
NA21089 NA21089 0       0       0       -9
NA21090 NA21090 0       0       0       -9
NA21091 NA21091 0       0       0       -9
NA21092 NA21092 0       0       0       -9
NA21093 NA21093 0       0       0       -9
NA21094 NA21094 0       0       0       -9
```

.fam File

- **The .fam file holds data on the individuals contained in the data set. Each line represents 1 individual and has six columns(tab delimiter)**

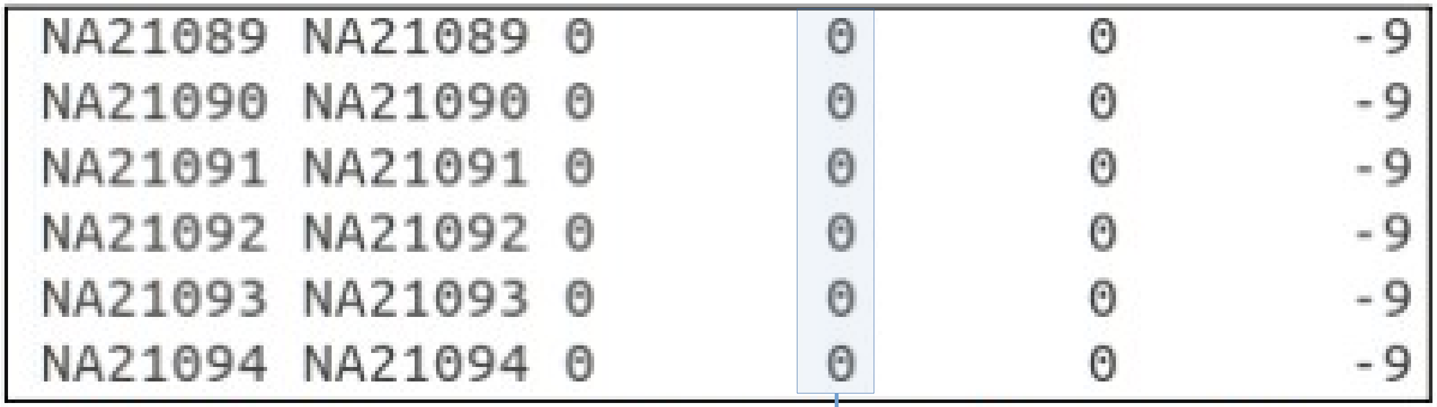


```
NA21089 NA21089 0        0        0        -9
NA21090 NA21090 0        0        0        -9
NA21091 NA21091 0        0        0        -9
NA21092 NA21092 0        0        0        -9
NA21093 NA21093 0        0        0        -9
NA21094 NA21094 0        0        0        -9
```

Father ID, default = 0

.fam File

- **The .fam file holds data on the individuals contained in the data set. Each line represents 1 individual and has six columns(tab delimiter)**

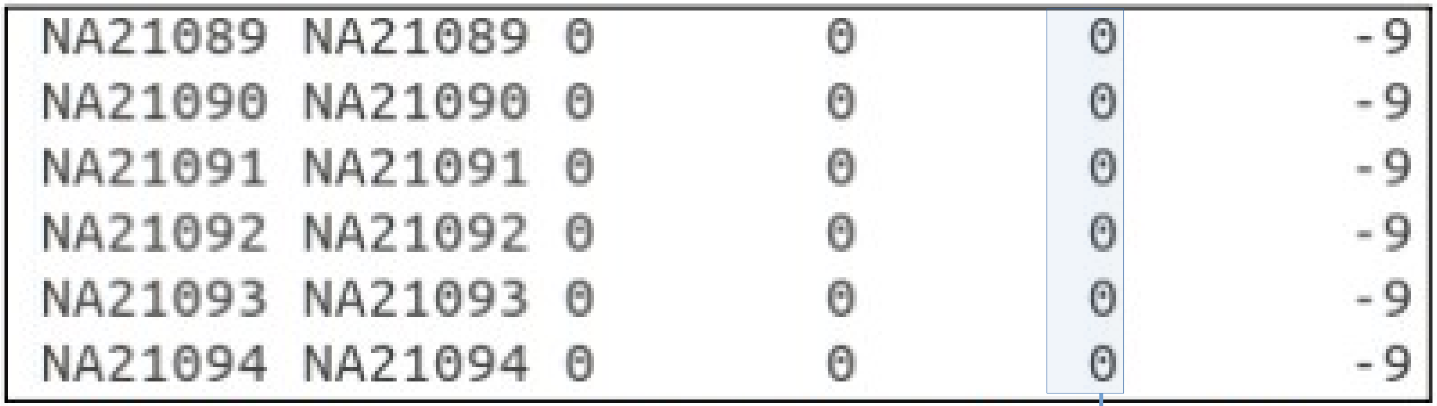


```
NA21089 NA21089 0       0       0       -9
NA21090 NA21090 0       0       0       -9
NA21091 NA21091 0       0       0       -9
NA21092 NA21092 0       0       0       -9
NA21093 NA21093 0       0       0       -9
NA21094 NA21094 0       0       0       -9
```

Mother ID, default = 0

.fam File

- **The .fam file holds data on the individuals contained in the data set. Each line represents 1 individual and has six columns(tab delimiter)**

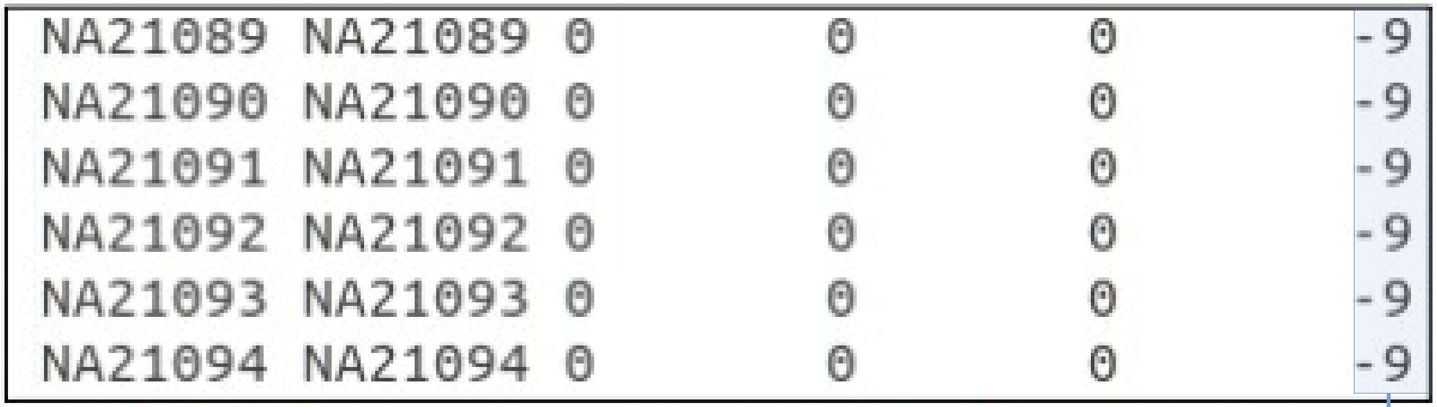


```
NA21089 NA21089 0        0        0        -9
NA21090 NA21090 0        0        0        -9
NA21091 NA21091 0        0        0        -9
NA21092 NA21092 0        0        0        -9
NA21093 NA21093 0        0        0        -9
NA21094 NA21094 0        0        0        -9
```

Sex, male = 1, female = 2, default = 0

.fam File

- **The .fam file holds data on the individuals contained in the data set. Each line represents 1 individual and has six columns(tab delimiter)**



```
NA21089 NA21089 0       0       0       -9
NA21090 NA21090 0       0       0       -9
NA21091 NA21091 0       0       0       -9
NA21092 NA21092 0       0       0       -9
NA21093 NA21093 0       0       0       -9
NA21094 NA21094 0       0       0       -9
```

Phenotype, control = 1, case = 2, default = -9

.bedp File

- **Binary with little-endian format, meaning the bits are read right to left within the byte.**
- **The first 3 bytes contain the header:** *1101100 00011011 00000001*
- **The remaining bytes in the file hold the phased haplotype data**

.bedp File

- **Phased Haplotype data held in 2 bits:**
  - 00 = homozygous for allele 1 (minor)
    - can represent null value as well
  - 01 = heterozygous, allele 1 is in the 1st position
  - 10 = heterozygous, allele 2 is in the 1st position
  - 11 = homozygous for allele 2 (major)

.bedp File

- **The file is a sequence of X blocks of N/4(rounded up) bytes, where X is the number of SNPs and N is the number of Individuals.**
  - ex. A data set of 14,207 SNPs and 300,013 individuals
    - 14,207(300,013/4) = 14,207(75,004) = 1,065,581,828
    - File size = 1,065,581,828 + 3 *(header)* = ~1.07GB

.bedp File

- The low-order(1st) bits of a block’s first byte stores the first individual’s haplotype data.
- The next two bits store the second individual's haplotype data, and so on for the 3rd and 4th individual.
- The second byte stores haplotype data for the 5th-8th samples, the third byte stores codes for the 9th-12th, etc.
- If N is not a factor of four, the extra high-order bits in the last byte of each block are always zero.

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 1st individual's 1st haplotype is 11, homozygous, allele 2 (major)

N / 4 (Rounded up) = Block Size

Individual's # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

1 / 4 = .25

.25 mod 1 = .25 (bits 0 and 1)

(1 – 1) * 2 + 3 + (1) = 4

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 2nd individual's 1st haplotype is "10", heterozygous, allele 2 in 1st position

N / 4 (Rounded up) = Block Size

Individual's # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

2 / 4 = .50

.50 mod 1 = .50 (bits 2 and 3)

(1 – 1) * 2 + 3 + (1) = 4

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 5th individual's 1st haplotype is "10", heterozygous, allele 2 in 1st position

N / 4 (Rounded up) = Block Size

Individual's # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

5 / 4 = 1.25

1.25 mod 1 =.25 (bits 0 and 1)

(1 – 1) * 2 + 3 + (2) = 5

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 6th individual's 1st haplotype is "11", homozygous, allele 2 (major)

N / 4 (Rounded up) = Block Size

Individual's # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

6 / 4 = 1.50

1.5 mod 1 = .50 (bits 2 and 3)

(1 – 1) * 2 + 3 + (2) = 5

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

Null Value

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 1st individual's 4th haplotype is "10", heterozygous, allele 2 in 1st position

N / 4 (Rounded up) = Block Size

Individual's # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

1 / 4 = .25

.25 mod 1 = .25 (bits 0 and 1)

(4 – 1) * 2 + 3 + (1) = 10

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 2nd individual's 4th haplotype is 11, homozygous, allele 2 (major)

N / 4 (Rounded up) = Block Size

Individual's # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

2 / 4 = .50

.50 mod 1 = .50 (bits 2 and 3)

(4 – 1) * 2 + 3 + (1) = 10

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 5th individual's 4th haplotype is 11, homozygous, allele 2 (major)

N / 4 (Rounded up) = Block Size

Individual's # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

5 / 4 = 1.25

1.25 mod 1 =.25 (bits 0 and 1)

(4 – 1) * 2 + 3 + (2) = 11

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

The 6th individual’s 4th haplotype is 10, heterozygous, allele 2 in 1st position

N / 4 (Rounded up) = Block Size

Individual’s # / 4 = X

X mod 1 = A

If A = .00, bits = 6 and 7
If A = .75, bits = 4 and 5
If A = .50, bits = 2 and 3
If A = .25, bits = 0 and 1

(SNP# - 1)* Block Size +3+(X (round up)) = byte location

6/4 = 2

6 / 4 = 1.50

1.50 mod 1 =.50 (bits 2 and 3)

(4 – 1) * 2 + 3 + (2) = 11

.bedp File

Example: 4 SNPs and 6 individuals

(4(6/4) + 3 = 4(2) + 3 = 11 bytes)

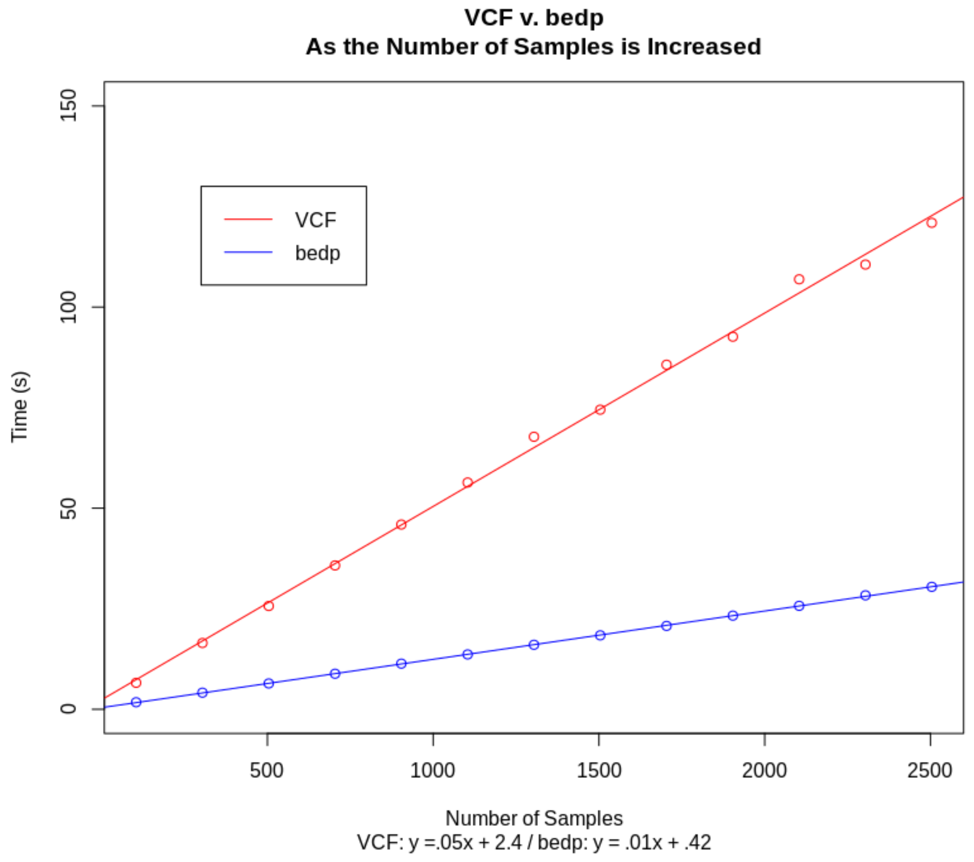
4th byte	5th	6th	7th	8th	9th	10th	11th
10101011	00001110	11111110	00001111	01010111	00000101	11111110	00001011

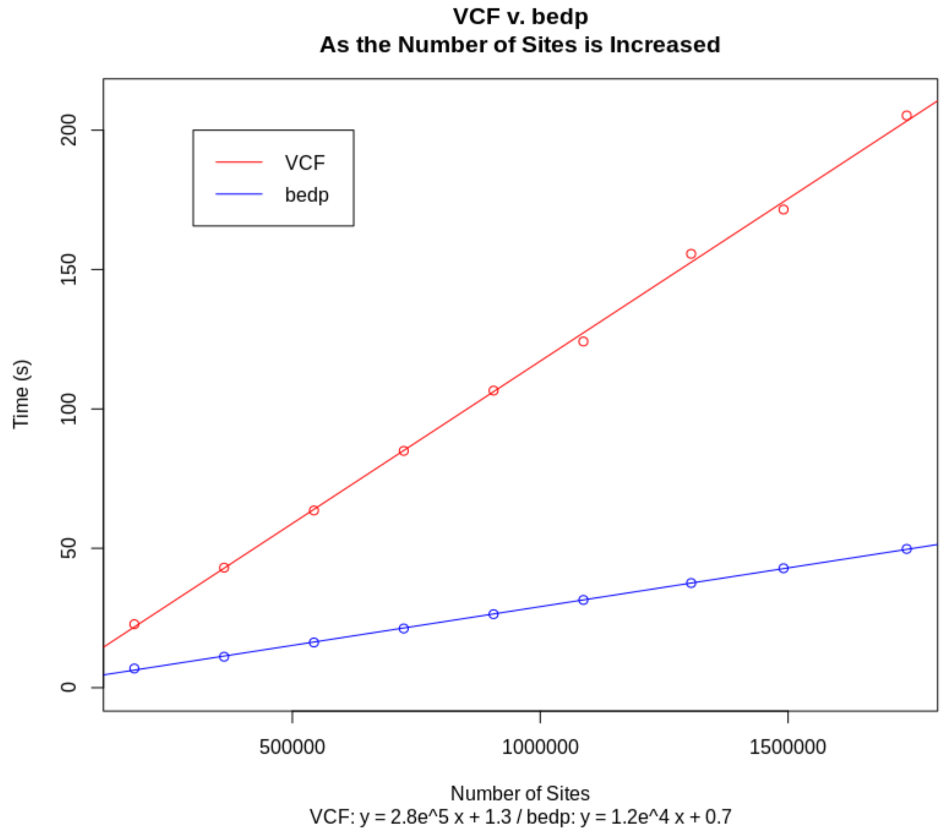
Null Value

Comparing BEDp to VCF using RaPID

- To compare the two file formats, VCF and bedp, RaPID was ran on 1000 genome project data, chromosome 21 and 22, and the run times for each format were observed
- To build a quality comparison, data sets of increasing sample size and increasing site size were created
- Sample size data sets used chromosome 22 and had sizes of 2504, 2304, 2104, 1904, 1704, 1504, 1304, 1104, 904, 704, 504, 304, and 104.
- Site size data sets used chromosome 21 and had sizes: 1739315, 1490841, 1304485, 1087070, 724712, 543534, 362356, and 181178.
- Using R, the graphs depicting run time performance were created and the line of best fit was found for each configuration.

Results





Using VCF, when a single sample is added to the 1000 genome project data it approximately adds .05 seconds to the running time of RaPID, where using bedp only adds approximately .01 seconds. A similar outcome is observed when sites are added to the 1000 genome project data. Using VCF an increase of approximately .00028 seconds is added and using bedp an increase of approximately .000012 seconds when a single site is added.

Converting to bedp

- **Using C++, a command line executable was created to convert files in the vcf.gz format into bedp**
- **vcf.gz requirements:**
  - Must be phased (haplotype separator = “|”, not “/”)
  - No missing haplotype values (there is no option in bedp)
  - Only the “GT” SNP data is converted

GitHub Repository: https://github.com/Ryan-J-Lewis/VCFtoBEDP

Using bedp with PLINK

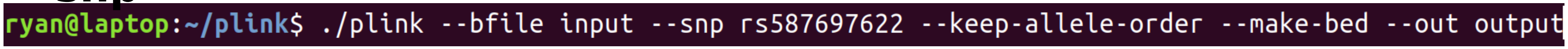
- PLINK is an open source whole genome association analysis toolset, designed to perform a range of basic, large-scale analyses in a computationally efficient manner. http://zzz.bwh.harvard.edu/plink/
- Currently only the --snp, --keep, and --remove, options have been tested and confirmed to work.
- To maintain data integrity the option,

  “--keep-allele-order” must be used

Using bedp with PLINK

- **Examples:**

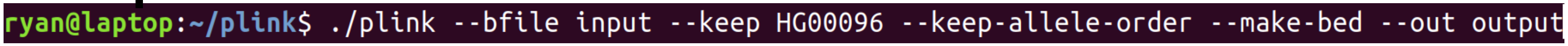
--snp



```
ryan@laptop:~/plink$ ./plink --bfile input --snp rs587697622 --keep-allele-order --make-bed --out output
```

This command will pull only the snp “rs587697622” for all individuals and store it in the files “output.bed”, “output.bim”, and “output.fam”

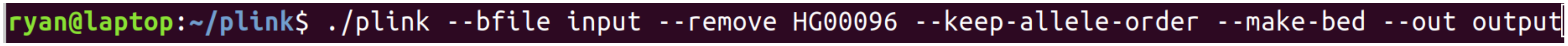
--keep



```
ryan@laptop:~/plink$ ./plink --bfile input --keep HG00096 --keep-allele-order --make-bed --out output
```

This command will pull all of the SNPs for the Individual “HG00096” and store it in the files “output.bed”, “output.bim”, and “output.fam”

--remove



```
ryan@laptop:~/plink$ ./plink --bfile input --remove HG00096 --keep-allele-order --make-bed --out output
```

This command will pull all individuals except “HG00096” and store it in the files “output.bed”, “output.bim”, and “output.fam”

Question?

ryan.j.lewis@uth.tmc.edu