Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Compression and fast retrieval of genomic wide genetic variants

Candidate
Denis Altomare

Advisor
Prof.ssa Barbara Di Camillo

Co-advisor
Dr. Francesco Sambo

March 10, 2015

Academic Year 2014/2015

Acknowledgements

I would like to thank Prof.ssa Barbara Di Camillo and Dr. Francesco Sambo for giving me the opportunity to work on this interesting topic and for leading me during the development of this thesis.
I would also like to thank Enrico e Matteo which are definitely the best flatmates ever and all the friends with whom I shared unforgettable moments in those university years.
I would like to thank Riccardo and all the people with whom I grew up during my life.
At last, I would like to thank my parents Fiorella and Mauro and my brother Luca which always supported me and pushed me into the right direction.
I feel lucky because I am surrounded by many awesome people, there are no sufficient words enough to say thank you.

> At lunch Francis winged into the Eagle to tell everyone within hearing distance that we had found the secret of life.
>
> *James D. Watson*

Contents

Abstract

The discovery of new sequencing technologies, with the consequent reduction of the cost of sequencing and the increasing interest on rare genetic variants, are pushing the research community towards the analysis of bigger and bigger datasets. Such data should be 1) compressed efficiently, in order to be storable occupying the minimum space and transmissible minimizing time, and 2) fast to retrieve, in order to minimizing data processing time.
For those reasons it is important to improve the performance of compression algorithms specifically designed to handle genetic data.
This thesis presents several improvements to SNPack 1.0, a state of the art algorithm for compressing and retrieving SNP data, designed for large scale association studies.
The algorithm has been improved in terms of compression time, compression ratio, loading time and type of data that can be handled. In fact, two of those improvements works on the compression algorithm leading the library to compress SNP data sparing a great amount of time and, at the same time, increasing the compression ratio. The lower loading time is a consequence of the smaller compressed data size.
The third of those improvements is an extension of the kinds of variants that can be compressed, in fact it has been designed and developed an algorithm for the compression of the other kinds of variants (such as multiallelic variants and structure variants) to provide a more complete library.
The new SNPack version is compared with several state of the art algorithms on the 1000G phase 3 release dataset, which is a detailed catalogue of human genetic variations consisting of 84801880 variants typed for a population of 2504 human individuals from 26 different populations.
A comparison with SNPack 1.0 exhibits a drastic reduction of the compression time and, simultaneously, an increment of the compression factor. This, indirectly affects the loading time, which is reduced because of the smaller file size.
Also the comparison with the other formats and tools highlights the superior performance of the new SNPack version for the compression on both the biallelic and the multiallelic variants.

Introduction

A genome-wide association study (GWAS) [Bush and Moore, 2012] analyses large sets of common and rare genetic variants in different individuals to find some significant correlation between some of those variants (typically SNPs) and a specific phenotypic trait (typically a disease).
The SNP (single nucleotide polymorphism) is the most common type of genetic variant and it consists in the mutation of a single nucleotide in the genome. New technologies for the sequencing of the DNA (the Next Generation Sequencing [Grada and Weinbrecht, 2013]) allow to easily produce huge amounts of GWAS data that can lead to the discovery of new associations between variants and traits.
The increasing amount of data brings to the pressing need for efficiently compression and fast retrieval of GWAS data. The popular genome wide analysis tool PLINK has introduced the binary PED (BED) format, which uses only two bits to store one genotype for one subject. This compression factor is not always sufficient, since the increasing dataset size can anyhow reach tens of Gigabytes on disk.
In the literature there are many algorithms for the storage of genome data of a small number of subjects [Brandon et al., 2009; Wang and Zhang, 2011; Christley et al., 2009]. Those methods usually store a reference genome and a map containing all the genetic variations and are not much efficient for the compression of GWAS data, which comprise bigger number of subjects and where the proportion of identical base pairs between subjects is lower.
Qiao et al. [2012] proposed SpeedGene which can compress biallelic SNPs. The genotype data is stored one SNP at a time and each of them is compressed using the best performing among three coding algorithms on the basis of the frequency of the SNP alleles. In fact, for each SNP, SpeedGene at first evaluates the space on disk required to store the SNP with each of the three compressing code, then it adopts the best performing one. The tool has been revealed good for compressing GWAS data and does not require the entire decompression to access a specific part of the genotype.
Sambo et al. [2014] proposed SNPack 1.0, a library based on an algorithm

which inherits and improves the SpeedGene compressing algorithms, furthermore it exploits the strong local similarity of the SNP data to ensure a more compact representation.
This thesis will present several improvements to SNPack 1.0 which make it more efficient and more complete.
In particular, those improvements lead the library to compress SNP data sparing a great amount of time and, at the same time, increasing the compression factor. Furthermore, the reduced compressed file size leads, indirectly, to a smaller loading time. All those improvements are produced without changing the file format, i.e. from a compressed file is impossible to know if it is built by SNPack 1.0 or by this new version.
In addition, it has been designed and developed an algorithm for the compression of the other kinds of variants (such as multiallelic variants and structure variants) to provide a more complete library.
The new SNPack version is compared with the state of the art algorithms on the 1000G phase 3 release dataset, which is a detailed catalogue of human genetic variations consisting of 84801880 variants typed for a population of 2504 human individuals from 26 different populations.
The obtained results show the outstanding performance of the new SNPack version. A comparison with SNPack 1.0 exhibits a drastic reduction of the compression time and, simultaneously, an increment of the compression factor. Also the comparison with the other formats and tools highlights the superior performance of the new SNPack version for the compression on both the biallelic and the multiallelic variants, in fact it performs better on each of the performance parameter took in consideration (which are compression time, loading time and storage space).
This thesis is composed of 5 chapters: Chapter 1 briefly summarizes the minimal biology notions which are essential to understand the mechanisms at the basis of the algorithms treated on this essay. Chapter 2 describes the main characteristics of the state of the art algorithms for compressing and retrieving genetic data, among which SNPack 1.0, the algorithm improved by the work described on this thesis. Chapter 3 illustrates the different improvements developed during this work. Chapter 4 describes the dataset adopted for the tests, what kind of tests has been conducted and what performance parameters are taken in consideration. At last, it analyses the results obtained by the tests.

Chapter 1

The DNA structure

This chapter briefly summarizes the minimal biology notions which are essential to understand the mechanisms at the basis of the algorithms treated on this essay. The information reported in this chapter are taken from [Neri and Genuardi, 2010] and [Allison, 2011], further reading on this topics may be found in the cited text as in any other genetic textbook.

1.1 The DNA

The *DNA* (acronym for deoxyribonucleic acid) is a molecule that encodes the genetic instructions used in the development and functioning of all known living organisms and many viruses.
It is a long chain of repetitive elements called nucleotides: each nucleotide is composed of:

- One pentose (which is a monosaccharide sugar with 5 atoms of carbon): the deoxyribose
- One phosphate group that gives the property of acid
- One *nucleobase* (or nucleotide base). The nucleobases are only four and are divided in two subcategories: the purines (*adenine* and *guanine*) and the pyrimidines (*cytosine* and *thymine*)

Therefore the DNA can be viewed like a very long string, unique for each living organism, where the characters are only four: A (adenine), G (guanine), C (cytosine) and T (thymine).
In the long chain of DNA can be identified sequences of three adjacent nucleotides called *codons*. Each codon can correspond to a single *amino acid* which is the basic element of the protein, or it can correspond to a starting

or stopping signal. In fact each protein starts with a start codon, followed by a sequence of amino acid codons and it ends with a stop codon.
DNA generically exists as two interwound strands, indeed two complementary strands in opposite directions are paired and bound by hydrogen bonds between the nucleobases. Adenine normally pairs with thymine and guanine pairs with cytosine. Due to this pairing, the basis lying on a strand are a sort of mirror image of the basis on the other strand hence the genetic informations carried by the two strands are the same.
The results of this pairing is the well known shape of the double helix [Watson et al., 1953].
As stated before, the DNA is a very long chain, indeed the cellular DNA can be many hundred million nucleotides long, as the human DNA is long more than 3 billion pairs. The number of base pairs *bp* is thereby used as a metric of DNA length but, due to the size of the DNA chain, in practice the unit of length is either the kilobase (*kb* or *kbp*) or the megabase (*Mb* or *Mbp*).

1.2 Genome organization

DNA is organized in *chromosomes* which are thread like structures composed mostly by proteins and DNA. Eukaryotic organisms such as human store the chromosomes in the the cell nucleus.
Human DNA has 23 pairs of chromosomes: one pair of sex chromosomes, which keeps most of the genetic traits linked to the sex of the individual and 22 pairs of autosomes, which contain all the other genetic hereditary information. Each chromosome is formed by non-coding and coding sequences (which are about only the 3% of the entire genome), the latter of the two are composed mostly by genes.
A *gene* can be considered the basic unit of heredity of a living organism, it holds the information to build and maintain an organism's cells and to pass genetic traits to the offspring. Every human has two copies for each gene, one inherited from each parent.
Most genes are the same for each people but a small part (less than 1%) changes from a person to another one and the different variants of those genes are called *alleles.*
The specific position of a gene (and of course, of an allele) on the chromosome is named *locus* (plural loci).
In a certain locus, each individual has two alleles: one belongs to the maternal chromosome and one belongs to the paternal chromosome. The information about which of the two alleles belongs to the maternal chromosome and which belongs to the paternal chromosome is called *phase.*

If, in a certain locus, the two alleles are different, the individual is said *heterozygous*, otherwise is said *homozygous*.
The complete set of genetic information in a organism is called *genotype*. This is to be distinguished from the phenotype, that is the organism's actual observable properties, such as morphology, development or behaviour.

1.3 Mutations

The 99,9% of the human genome is the same for each person. In the remaining 0.1% lies the genetic code which is responsible for the trait differences between individuals. Therefore, on average between two people one base pair over one thousand is different and, because the DNA base pairs are more than 6 billions, between two people there are more than 3 million different base pairs.
Those differences are caused by some genetic variations which can be classified in this way:

- Small scale mutations: mutations that affect a gene in one or few nucleotides. Those mutations are subdivided in:
    - Point mutations: this kind of mutations affect one single nucleotide. A point mutation occurring in a protein coding region can be classified in three type, depending upon what erroneous codon codes for.
        - Silent mutations: it codes for the same amino acid. In this case the coded protein will be the same.
        - Missense mutations: it codes for a different amino acid.
        - Nonsense mutations: it codes for a stop codon and can truncate the protein.
    - Insertions: in this case one or more nucleotides are added in the DNA.
    - Deletions: one or more nucleotides are deleted from the DNA.
- Large scale mutations: those mutations involve the chromosomal structure.
    - Amplifications: duplications of an entire chromosomal region, leading to multiple copies of all the genes in these regions
    - Deletions: deletions of a large chromosomal region, leading to loss of the genes within those regions

- Mutations: whose effect is to juxtapose previously separate pieces of DNA.
    - Chromosomal translocations: interchanges of genetic parts from non-homologous chromosomes.
    - Interstitial deletions: an intra-chromosomal deletion that removes a segment of DNA from a single chromosome, thereby apposing previously distant genes.
    - Chromosomal inversions: reversing the orientation of a chromosomal segment.
- Loss of heterozygosity: loss of one allele in an organism that previously had two different alleles.

1.3.1 Single Nucleotide Polymorphism

A *single nucleotide polymorphism* (SNP, pronounced snip) is a point mutation occurring commonly within a population (e.g. 1%) in which a single nucleotide differs between individuals of the same species. Theoretically, a SNP could have four different alleles, one for each nucleotide, but in practice most of the SNPs have only two different alleles. This means that, in this case, the feasible configurations are three if the phase is not considered (homozygous of the first allele, heterozygous and homozygous of the second allele) and four if the phase is considered (because there are two different feasible heterozygous configurations).

Within a population, SNPs can be assigned a *minor allele frequency* (MAF) which is the lowest allele frequency at the SNP's locus that is observed in this particular population. This kind of mutation is the most common, indeed it's estimated that the 95% of all the mutations are SNPs [1000 Genomes Project Consortium, 2012].

A *genome wide association studies* (GWAS) analyses large sets of genetic variants (mostly SNPs) in different individuals in order to find some significant correlations between haplotypes and a specific phenotypic trait (typically a disease). *Haplotypes* are patterns of sequence variation, like stretches of continuous DNA containing a specific set of alleles.

SNPs do not necessarily cause disease, but they can help to determine the likelihood that someone will develop a particular disease.

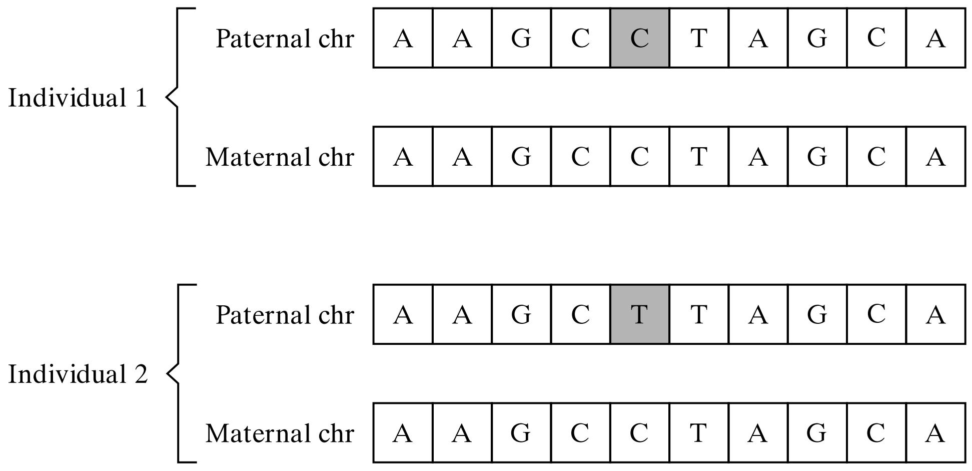


Figure 1.1: A single nucleotide polymorphism example. In this example the difference between the two DNA fragments consist of a single nucleotide.
In a certain locus, the individual 1 has a homozygous genotype, because it presents the nucleobase C in both the copies of the chromosome. In the same locus, the individual 2 has a heterozygous genotype of the allele C and T.

1.4 Linkage disequilibrium

Linkage disequilibrium (LD in shorter) measures the degree to which alleles at two loci are associated.
The explanation of this phenomenon is clearer with an example.
Let us take into consideration two biallelic loci A and B, with alleles A_1, A_2 and B_1, B_2, respectively. Let x_{ij} be the frequencies of the individuals with both A_i and B_j alleles in the population and p_i and q_j be the frequencies of individuals with alleles A_i and B_j, respectively (see Table 1.1a and 1.1b). If the two loci A and B are independent from each other, the frequency of the haplotype A_iB_j should be equal to the product p_iq_j, as shown in Table 1.1c (that is $x_{ij} = p_iq_j$).
Then, let $D = x_{11} - p_1q_1$, if $D = 0$ there is linkage equilibrium, which means that the loci are independent from each other. Otherwise there is linkage disequilibrium: if $D > 0$ the haplotype A_1B_1 is overrepresented, on the other hand if $D < 0$ the same haplotype is subrepresented.
Therefore in a population, SNPs in high LD have identical genotype for the majority of subjects.
Linkage disequilibrium is a consequence of the *crossing over*, a process in which two homologous chromosomes exchange some genetic code. The two

Haplotypes	**Frequencies**
A_1 B_1	x_{11}
A_1 B_2	x_{12}
A_2 B_1	x_{21}
A_2 B_2	x_{12}

(a) Haplotype frequencies based on the population.

Allele	**Frequencies**
A_1	$p_1 = x_{11} + x_{12}$
A_2	$p_2 = x_{21} + x_{22}$
B_1	$q_1 = x_{11} + x_{21}$
B_2	$q_2 = x_{12} + x_{22}$

(b) Allele frequencies based on the population.

	$\mathbf{p_1}$	$\mathbf{p_2}$
$\mathbf{q_1}$	p_1q_1	p_1q_1
$\mathbf{q_2}$	p_1q_2	p_2q_2

(c) Haplotype frequencies for independent loci.

Table 1.1: Linkage disequilibrium. Let us take into consideration two biallelic loci A and B, with alleles A_1, A_2 and B_1, B_2, respectively. Let p_i and q_j be the frequencies of the alleles A_i and B_j in the population, respectively, If the frequency x_{ij} of the haplotype A_iB_j is different to the product p_iq_j, the two loci are in linkage disequilibrium, otherwise they are in linkage equilibrium.

homologous chromosomes break at the same locus and then exchange a piece of DNA reconnecting to the different end piece.
If two genes are close together on a chromosome, the likelihood that during a crossing over these two genes will be separated is lower than if they were farther apart. The closer together are the loci took in exam, thus the higher is the linkage disequilibrium degree.
The DNA is known for exhibiting regions with strong similarity which are called *LD blocks*, separated by small regions in linkage equilibrium [Wall and Pritchard, 2003].

Chapter 2

State of the art

Several different standards for compressing genetic variations data are present in the literature.
Those standards have different characteristics, they are differentiated by what kinds of variants they can store (i.e. only biallelic SNPs rather than several different kinds of variations) and how they store the the data (i.e. some standards store data in plain text, other standards store the substantial part of the data in binary form).
The key aspects on which we mainly focus on are two: 1) the kind of variants stored with those standards and 2) the performance achieved adopting those standards (obviously storing data in binary form permits to save a significant quantity of disk space but many types of compression require a complete decompression during the access to the data, this means long time to load in memory).
This chapter will briefly describe the most used formats, the last of those is SNPack which is the standard targeted by the improvements treated in this thesis.

2.1 VCF (Variant Call Format)

The Variant Call Format [Danecek et al., 2011] is an uncompressed text-based format which can store all kinds of variants in an unique file. This format is currently one of the most adopted standards to store the GWAS data and its most recent version is 4.2.
The VCF files are formed by two parts: the first part is composed by meta-informations and the second part is composed by the data (the body).
The meta-informations are represented by lines starting with a ## and are placed on the top of the file. This part contains informations like the vcf file

format and the descriptions of the fields which will be used in the body.
The body is formed by one line for each variant present on the dataset. It starts with a standard header line which names eight mandatory fields that contain informations like chromosome, position, variant id, reference allele, alternative alleles, other variant's informations and, if present, genotype of the individuals.
This kind of representation is very flexible, indeed it can store SNPs, indels (either biallelic or multiallelic) and large structure variants.
VCF is a pure text file format, hence the data can be easily read with a simple text editor, but the disadvantage is the huge file size. For this reason the VCF files are often compressed with a general purpose tool. Further details can be found in [Danecek et al., 2011].
The following is a simple VCF file toy example which contains three variants (all of them belong to the chromosome 22) and two subjects.

```
##fileformat=VCFv4.1
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT HG00096 HG00097
22 16050075 . A G 100 PASS . GT 0|0 0|0
22 16050678 rs139377059 C T 100 PASS . GT 0|0 0|0
22 16053659 rs141264943 A C 100 PASS . GT 0|0 1|0
```

Note that the first SNP does not present the ID, this means that no identifier is available for that variant. The genotype is encoded like $m|n$ (where m and n are non negative integers), the 0 represents the reference allele, the 1 represents the first alternative allele, 2 represents the second alternative allele (if present) and so forth. In this thesis we call *VCF code* the integer number which represents the variant allele (m and n). For example, the first subject of the third SNP is homozygous with allele A, while the second subject is heterozygous with alleles A and C.

2.1.1 BCF2

BCF2 [Danecek et al., 2011] is the binary counterpart of the VCF file format. A BCF2 file is composed of a mandatory header, followed by a series of BGZF compressed blocks of BCF2 records. The BGZF is a GZIP variant, it is bigger than a GZIP traditional file but it is faster for random access. The BCZF blocks allow this kind of files to be indexed with tabix [Li, 2011] which is a generic tool that indexes position sorted files in TAB-delimited formats and quickly retrieves features overlapping specified regions.
For efficiency reason only a subset of VCF is supported by BCF2.
To manipulate and get informations from VCF and BCF files the standard software is the widely adopted program package vcftools, which is composed

by some modules written in perl and some binary executable. Recently, bcftools has been developed, this new tool presents most of the functionalities offered by vcftools but with better performance.

2.2 PLINK

PLINK [Purcell et al., 2007] is a widely used tool for genome wide association analysis. It has two different versions of file format: one to store data completely in a text-based format (on files .ped/.map) and the other one to store the consistent part of the data in a binary format (on files .bed/.bim/.fam). Both file types are designed to compress biallelic SNPs (which are the most common variants in the genome by far).
The following sections describe the two file formats.

2.2.1 The text-based file format

The .map/.ped PLINK format is composed of two text files and it stores the genotype in an uncompressed manner.

- **.map:** this text file contains all the information about the genetic markers.
  It is composed of four columns which are: chromosome, SNP ID, genetic distance (morgans) and base pair position (bp units). Each line corresponds to a single marker.
  The following is a simple MAP file (there are three SNPs):

  ```
  1  rs123456  0  1234555
  1  rs234567  0  1237793
  1  rs233556  0  1337456
  ```

  It contains three markers belonging to the chromosome 1, each of them has an unique identifier and it has the genetic distance set to 0 (which is missing value). The last value is the base pair position in the chromosome.

- **.ped:** this text file contains the information about all the individuals (one per line), including their genotype.
  It is a white spaced file containing six mandatory columns which are: family ID, individual ID, paternal ID, maternal ID, sex and phenotype. These columns are followed by the genotype, which is encoded with a

character (all the markers should be allelic).
The following is a simple PED file (there are two individuals and three SNPs):

```
FAM001  1  0 0  1  2  A A  G G  A C
FAM001  2  0 0  1  2  A A  A G  0 0
```

This example shows two subjects, with the same family ID (but different individual ID), the same sex, the same phenotype and missing paternal and maternal ID. For each subjects three SNPs are recorded, the last of which is missing for the second individual.
This file type is completely text-based, hence for big dataset that means long loading time and big data sizes.

2.2.2 The binary file format

The binary counterpart of the PED file type is the binary PED (BED).
It is composed of two text files which contain the basic information about the individuals and the SNPs, plus a binary file containing all the genotype in a compressed format:

- **.fam:** this text file contains all the information about the individuals. It is composed by the firsts six PED file columns which are: family ID, Individual ID, paternal ID, maternal ID, sex and phenotype. Each line corresponds to one individual.

- **.bim:** this text file contains all the informations about the SNPs. This is an extended MAP file, it contains the MAP file's four columns plus two extra columns which globally are: chromosome, SNP ID, genetic distance (morgans), base pair position (bp units), first allele name, second allele name. Each line corresponds to one SNP.

- **.bed:** this binary file contains all the genotype in a compress format. The header is composed by three fixed bytes, the remaining of the file is the genotype data. Each genotype is encoded in two bits, 00 stands for first allele homozygous, 01 stands for heterozygous, 11 stands for second allele homozygous and 10 stands for missing genotype.

The binary PLINK file format compresses all the genotype data with a good compression ratio (with respect to the plain text format) and permits the access to a single genotype without the complete decompression of the data. Further details on the PLINK tool can be found in [Purcell et al., 2007].

2.3 SNPack

The SNPack algorithm [Sambo et al., 2014] is designed for the compression and the fast retrieval of biallelic SNPs. It exploits the SNPs Minor Allele Frequency (MAF) and the strong local similarity typical of SNP data (linkage disequilibrium).
The compression is decomposed in two tasks: first the linkage disequilibrium blocks are detected and summarised, in terms of differences with a common nearby reference SNP, then the reference SNPs are compressed efficiently exploiting the information on their MAF.
When compressing an LD block, thus, it suffices to store one of the SNPs in its entirety and use it as a reference to summarise the other SNPs in the block, storing just their variations with respect to the common SNP. The former is named *reference SNP* and the latter *summarised SNPs*.
The compression of the reference SNPs is performed using the best performing among five compression algorithms, or codes, which are an improvement of the SpeedGene's sub-algorithms [Qiao et al., 2012].
This algorithm stores the genotype data in a binary format and does not require the full decompression of the data prior to the access to a single genotype. The information about the individuals and the SNPs is stored in the PLINK file format .fam and .bim respectively. All the details and the tests on SNPack can be found in [Sambo et al., 2014]. The compression codes, used to store the reference and the summarised SNPs, are described in the following subsection.

2.3.1 The compression codes

The reference SNPs are stored using the best performing among those five code:

- The **code 1** encodes the genotype of a subject in a two digits number representing the number of copies of the minor allele (which can be 00, 01, 10 to represent 0, 1 or 2 copies) or the missing genotype (11).

- The **code 2** is designed for SNPs with low MAF and records only the subjects with the heterozygous and rare homozygous genotype, plus the ones with missing genotype. For each subject category (rare homozygous, heterozygous and missing), rather than storing each subject index, the first index is stored, followed by the differences between all pairs of consecutive indices. In this way, only $bdiff_{aA}$, $bdiff_{aa}$ and $bdiff_{miss}$ bits are required to represent both the first index and a difference between consecutive indices of rare homozygous, heterozygous and missing genotype, respectively.

- The **code 3** is designed for SNPs exhibiting a large number of heterozygous subjects. It uses a binary array of n digits to indicate the subjects with the homozygous genotype. If the jth subject has a heterozygous genotype then a 1 is put in position j.
  The indices of the missing and the rare homozygous genotypes are recorded in the same way adopted in the code 2.
- The **code 4** is similar to code 2 but swaps the role of the missing value with the one of frequent homozygous subjects. This works well if there is a majority of missing values.
- The **code 5** is similar to code 3 and swaps the role of the missing values with the one of heterozygous subjects.

The compression of the summarized SNPs is performed using the **code 6**, which store the index distance from the reference SNP and the genotype variations with respect to the reference SNP.
A descriptive image of the codes composition can be viewed in Figure 2.1. As shown by the figure, each code starts with the code ID stored in three bits. Code 1 ignores the subsequent five bits, then stores the genotype of 4 subjects for each Byte.
Code 2 exploits the last 5 bits of the first Byte and the two subsequent Bytes to store bdiff_{aA}, bdiff_{aa} and bdiff_{miss}. The code, then, stores the number of rare homozygous, heterozygous and missing subjects, each requiring $\lceil \log_2(n) \rceil$ bits, followed by, for each of the three set of indices, the first index and the differences between pairs of consecutive indices.
Code 3, similarly to the previous code, stores bdiff_{aa} and bdiff_{miss} into the first two Bytes, followed by the number of rare homozygous and the number of missing genotypes and by the two sets of indices. Code 3 ends with a binary array of n digits, where a 1 at the ith position means the ith subject has an heterozygous genotype.
Code 4 is similar to code 2 but it stores explicitly the indices of frequent homozygous, rare homozygous and heterozygous.
Code 5 is similar to code 3 but it stores explicitly the indices of rare homozygous and heterozygous, exploiting the binary array of n digits to indicate the missing genotypes.
Code 6, after the code ID, records the index of the reference SNP by storing the index distance using 1 bit for the sign (upstream or downstream) and 20 bits for the distance. Then it stores the number of genotype variations with respect to the reference SNP followed by the list of their indices and values.

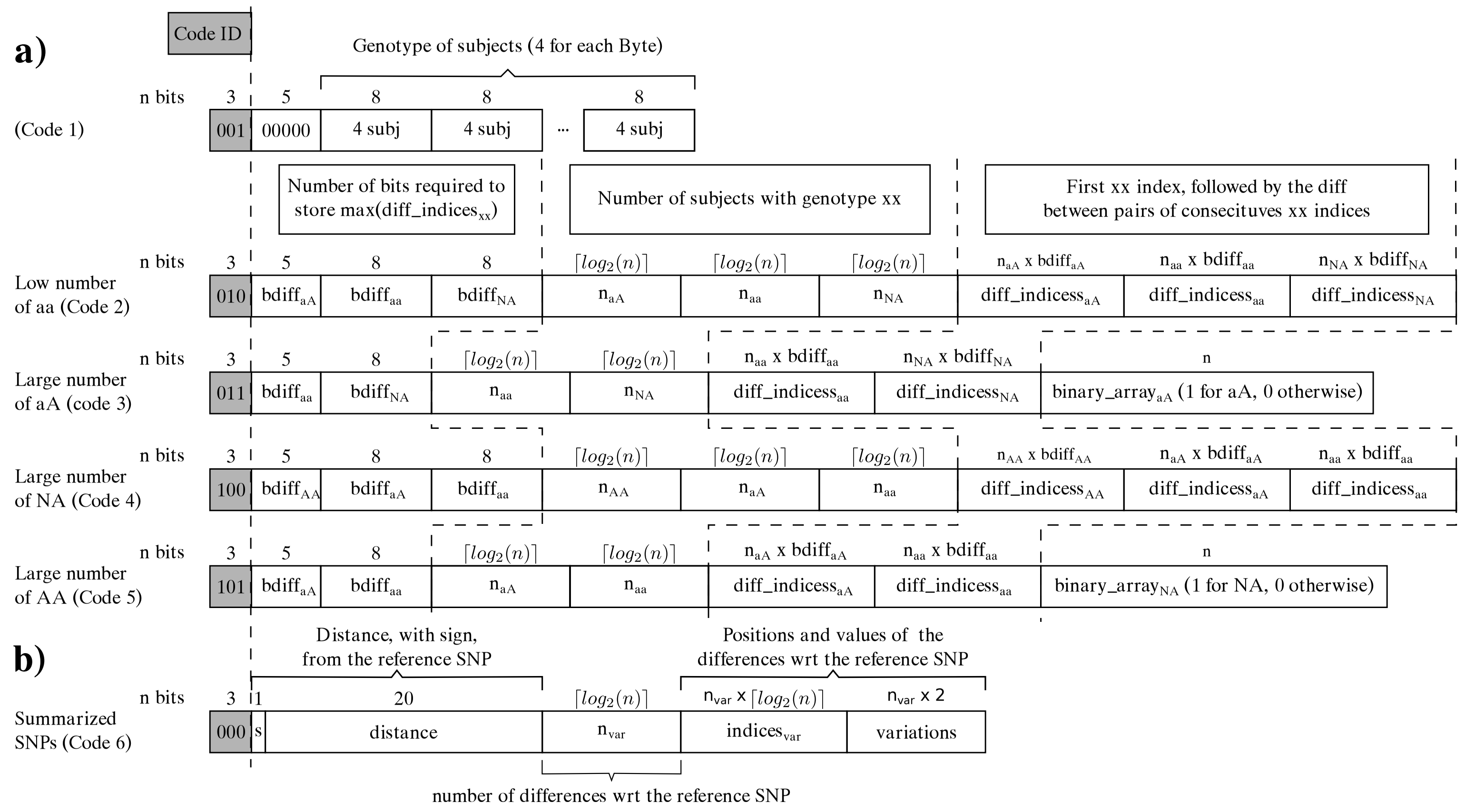


Figure 2.1: Schematics of the 5 codes for compressing the reference SNP (a), plus the code for compressing the summarized SNPs (b). For each code, we report the sequence of stored values and, on top of them, the number of allocated bits.

2.3.2 The compression algorithm

As said at the end of the Section 1.4, in a population SNPs in high LD have identical genotypes for the majority of subjects, therefore it is sufficient store one SNP and use it as a reference to summarise the other SNPs in the LD block, storing just the differences with respect to the common SNP. The former is called *reference SNP* and the latter *summarised SNP*.
The algorithm (whose pseudocode is reported in Algorithm 1) starts by computing the best of the 5 codes for each SNP and the corresponding cost in Bytes (the cost to store one variant is simply the space required to store it). Then, for each SNP i, the algorithm searches its neighbourhood $N[i]$, which consist of all the SNPs j that can be effectively summarised by i. To do this, it computes *partial gain*$[i, j]$ that is the subtraction between the cost of summarising j with i and the cost of storing j.
All the SNPs j with positive partial gain are added to the set of useful neighbourhoods of i, $uN[i]$, and the sum of their gain constitute *gain*$[i]$ using i as reference SNP for summarising all the SNPs in the $uN[i]$.
Once the gain of all the SNPs has been computed, the algorithm builds up the summary, either assigning to a SNP the role of reference or the role of summarized by indicating which reference should summarised it, in a greedy fashion.
The algorithm, in fact, selects iteratively the SNP m with the highest gain and sets it as reference of all the SNPs in its useful neighbourhood, which are set to summarised SNPs.
Then it sets to 0 the gain of m and, for each SNP j in $uN[m]$, it sets to 0 the gain of j and it updates all the gains of the SNPs which have j in their useful neighbourhood (because a SNP can be summarised by only one reference).
The depth of the summary is limited to 1 (it could be not limited), in fact one summarised SNP cannot be a reference SNP for other SNPs. This design choice is meant to limit the time complexity of the decompression phase, which is described in the Subsection 2.3.4.
Finally, the data is written to disk using one of the 5 aforementioned codes to store the reference SNPs and a sixth code (named Summarised), to store the summarised SNPs.
The computational complexity of the algorithm is dominated by two operations: 1) the computation of the genotype variations between each SNP and its neighbours and 2) the iterated computation of the maximum element of the vector gain. If n is the number of subjects, p is the number of SNPs and $|N|$ is the number of SNPs in a neighbourhood, the former operation has complexity $O(pn|N|)$ and the latter $O(p^2)$.
To limit the complexity of computing genotype variations and to cope with

the variable patterns of linkage disequilibrium throughout the genome, a heuristic procedure (described in the Subsection 2.3.3) is adopted for adaptively setting the size of the neighbourhood of a SNP. The limitation of the neighbourhood size, when possible, limits the complexity factor.
Since the LD on the border of the chromosomes is not expected, this compression algorithms is thought to be run separately on each chromosome. Moreover, to take advantage of the parallelism of the compression task, the chromosome is split into several chunks of equal size (the number of which can be chosen in input) and each chunk is compressed by a different thread.

Algorithm 1 The SNPack compression algorithm

```
1: Load SNP data
2: for all SNP $i$ do
3:     $cost[i] \leftarrow$ Byte size of the best code for $i$
4: end for
5: for all SNP i do // useful neighbours, i.e. SNPs worth being summarised by i
6:     $uN[i] \leftarrow \emptyset$
7:     for all SNP $j \in N[i]$ do // Neighbourhood of $i$
8:         Compute genotype variations between $i$ and $j$
9:         $pg[i,j] \leftarrow cost[j]$ - cost of summarising $j$ with $i$ // partial gain
10:        if $pg[i,j] > 0$ then
11:            $uN[i] \leftarrow uN[i] \cup j$
12:        end if
13:    end for
14:    $gain[i] \leftarrow \sum_{j \in uN[i]} pg[i,j]$
15: end for
16: $m \leftarrow argmax_i(gain[i])$
17: while $gain[m] > 0$ do
18:     $summary[uN[m]] \leftarrow m$
19:     $gain[m] \leftarrow 0$
20:     $gain[uN[m]] \leftarrow 0$
21:     for all $j \in uN[m]$ do
22:         for all $k$ such that $j \in uN[k]$ do
23:             $gain[k] \leftarrow gain[k] - pg[j,k]$
24:         end for
25:     end for
26:     $m \leftarrow argmax_i(gain[i])$
27: end while
28: Write data to disk according to $summary$
```

2.3.3 The JIT heuristic

The JIT heuristic procedure is designed for adaptively setting the size of the neighbourhood of a SNP, which permits to limit the complexity of computing genotype variations and to cope with the variable patterns of linkage disequilibrium throughout the genome.
This procedure (whose pseudocode is reported in Algorithm 2) incrementally grows the neighbourhood, starting from a neighbourhood of 10 SNPs and doubling the neighbourhood size if a new useful neighbour SNP is found among the newly added SNPs in the neighbourhood. Thus, the size growth of the neighbour stops if no new SNPs are added to the useful neighbourhood in an iteration or if a maximum extension w (received as input from the user) is reached.
This algorithm prevents unprofitable computation in that zones of DNA with low LD degree.

Algorithm 2 JIT heuristic used in the SNPack original version

```
1: for all SNP $i$ do
2:     $jit \leftarrow 10$
3:     $oldjit \leftarrow 0$
4:     $improved \leftarrow true$
5:     while $improved = true \wedge oldjit < w$ do        // $w$ is the max neighb. size
6:         $improved \leftarrow false$
7:         for all $j \in [i - jit, i + jit] \setminus [i - oldjit; i + oldjit]$ do
8:             $pg[i, j] \leftarrow compute\_pg(i, j)$
9:             if $pg[i, j] > 0$ then
10:                $improved \leftarrow true$
11:                add $j$ to the $i$'s useful neighbourhood
12:                $gain[i] \leftarrow gain[i] + pg[i, j]$
13:            end if
14:        end for
15:        $oldjit \leftarrow jit$                              // Increase jitter
16:        $jit \leftarrow min(2 \cdot jit, w)$
17:    end while
18: end for
```

2.3.4 Decompression

The decompression is made on a chromosome at a time: initially the entire compressed chromosome is loaded, then with a first scan the algorithm locates and decodes all the reference SNPs (which are encoded with the firsts five codes). With a second pass the decompression algorithm reconstructs all the summarized SNPs (using the the information about the reference SNPs which are all already decoded).
These two passes are sufficient to decode all the genomic data because of the summary depth limited to 1. If the summary depth were not limited to 1, more than two passes could be necessary to decompress all the data, leading to a slower decompression phase.

Chapter 3

Methods

One of the SNPack's strengths is the extremely short loading time, which is guaranteed by the unitary summary depth level.
This algorithm performs very well even for what concerns two other important points: it is characterized by a high compression factor and a short compressing time. Those results are achieved due to the use of the 5 codes and to the exploitation of the local similarities typical of the SNP data.
Nevertheless, there are some points in which SNPack can be improved. According to Sambo et al. [2014] the core of the compression algorithm is the choice of which SNP should be coded as reference and which SNP should be summarised by each reference SNP. In general, the problem is NP-complete and the adopted solving strategy is based on greedy choices, which give satisfactory results in short time. Actually, both the compression factor and the compressing time are characteristics of the utmost importance.
As will be shown in the following sections the compressing time, although short, is improvable.
The fact that SNPack is designed to compress only biallelic SNPs, even though those are the most common variants by far, is a weakness.
For this reason, a new compression technique has been developed to store the other kinds of variants, in order to provide a more complete tool.

3.1 Optimization of the summarized SNPs

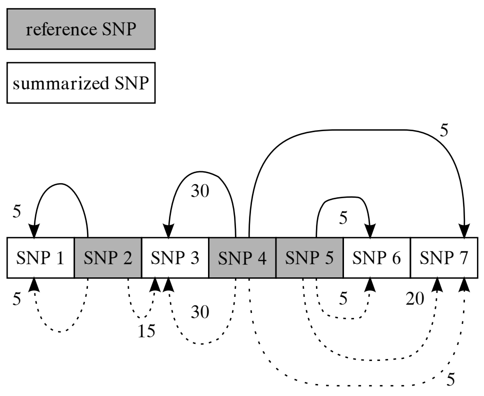
A simple algorithm which works on the allocation of the SNPs as reference and as summarized SNP has been developed.
This algorithm starts from a feasible solution, thus it assumes that all the SNPs are already allocated either as reference or as summarized SNP (and, in the latter case, it is already recorded which reference SNP summarises it).

The purpose of this algorithm is to find the optimal summary once the status of reference SNP and summarized SNP are fixed. All the SNPs are still either reference SNP or summarized SNP as they were before but, for each summarized SNP s, the algorithm searches for the best reference SNP that summarizes it effectively (i.e. the reference SNP that can summarize s producing the higher gain).
The algorithm is applied after the compression algorithm, when the role of reference or summarised is assigned to all the SNPs. As shown by the pseudocode in Algorithm 3, the algorithm, for each summarised SNP s, searches the reference SNP r which can summarises s obtaining the highest gain. After that, if the reference SNP r can summarise s better than the previous one, then r became the reference SNP of s (as shown in lines $4-6$).
To do this, the algorithm performs approximatively a number of comparison equal to the size of a neighbourhood for each summarized SNP, hence the computational complexity is $O(p|N|)$ where p is the number of SNPs and $|N|$ is the size of a neighbourhood.
The toy example reported in Figure 3.1 shows exactly the behaviour of this

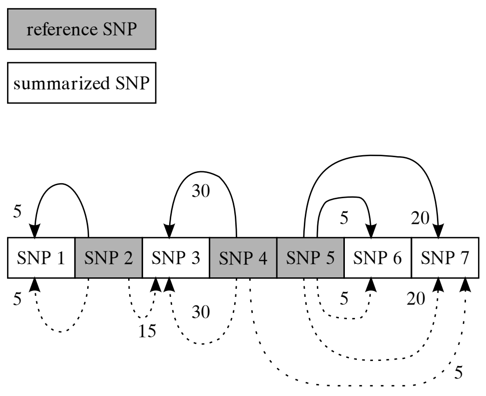
Algorithm 3 Optimize summary

```
1: for all summarized SNP $s$ do
2:     $r_{actual} \leftarrow$ the reference SNP of $s$
3:     for all reference SNP $r$ that could summarize $s$ do
4:         if $r$ summarizes $s$ better than $r_{actual}$ then
5:             $r$ became the reference SNP of $s$
6:             $r_{actual} \leftarrow r$
7:         end if
8:     end for
9: end for
```

algorithm. In fact, the status of reference and summarised SNPs remains unaltered, but the allocation of the SNPs changes (in this example one allocation is modified). SNP 7, which was summarised by SNP 4, is summarised by SNP 5 after the optimization algorithm, with a gain of 20 instead of 5. This simple and artificially crafted example shows one single change which increases the total gain from 45 to 60.



(a) Situation before the optimization.



(b) Situation after the optimization.

Figure 3.1: This is a toy example to explain the effects of the optimization. The white rectangles represent summarized SNPs and the shaded rectangles represent reference SNPs. The straight arrows start from a reference SNP and end to a summarized SNP and indicate that the reference SNP summarises the summarized SNP. The dotted arrows start from a reference SNP and end to a summarized SNP and indicate that the reference SNP could summarise the summarized SNP with a positive gain. The arrow's label is a number which indicates the gain obtained using that reference SNP to summarise that summarized SNP.
The left figure shows a situation similar to which can occur at the end of the compression algorithm. The right figure shows how the optimization algorithm change the situation in this case.

3.2 Speed up of the gain computation

The greedy allocation of the SNP as reference or as summarized is composed of two main phases.
In the first phase, the useful neighbourhood for each SNP and its related gain are computed. The second phase is composed of the greedy iterations until all the SNPs are marked as reference or as summarized (see Subsection 2.3.2). Recalling what said in the previous chapter, in SNPack 1.0, the first phase is done comparing each SNP i with all the SNPs in its neighbourhood.
To limit the computational complexity, the neighbourhood size is incrementally increased, starting from a dimension of 10 SNPs and doubling its size if new SNPs are added to the useful neighbourhood. The growth of the neighbourhood size is stopped either if no SNPs are added to the useful neighbourhood or if a maximum size w is reached (the algorithm's pseudocode is shown in Algorithm 2).
The enhancement described in this subsection has the purpose of drastically lower the time spent during the first phase of the compression algorithm (with a more incisive effect than the JIT heuristic).

The proposed improvement works mainly on the first phase and requires another parameter besides the maximum neighbourhood size w. The new parameter is a real number $0 < r <= 1$, and it defines a small neighbourhood of size rw.
The computation of the useful neighbourhood and the gain of each candidate reference SNP, during the first phase, is made on this small neighbourhood of size rw (which is smaller than the neighbourhood of size w adopted in SNPack 1.0).
The change on the second phase is small but very important. In fact, when a SNP is chosen to be a reference SNP, the entire neighbourhood of maximum size w is examined and all the SNPs which can be positively summarized are added to the useful neighbourhood (while in SNPack 1.0 only the SNPs in the useful neighbourhood became summarised SNP).
Except for this variation, the second phase remains unaltered.
This version of the algorithm makes the SNP choice more greedy than the original version, in fact the choice is based on a lower amount of information. The procedure, however results much faster (as shown in the results in Chapter 4). In fact, the entire exploration of the neighbourhood is done only for those SNP which will surely be reference SNP, unlike in the original version where this exploration is performed for all the SNPs, saving in this way a great amount of time.
The purpose of this enhancement is to drastically lower the time spent during the first phase. For this reason, it makes the JIT heuristic not essential, indeed the adoption of that is suggested only to reach the shortest compression time (in spite of the compression ratio).
The compression algorithm workflow of the new SNPack version (which includes the improvements described in the Section 3.1 and in the Section 3.2) is reported in the Algorithm 4. As shown by the pseudocode, first the data is loaded, then the algorithm compress the data (the compression algorithm is composed of two main phases) and, at last the algorithm performs the summary optimization which improves the quality of the compression.

Algorithm 4 Compression algorithm workflow of the new SNPack version

```
1: Load data to be compressed
2: First phase: computation of the candidate reference SNPs gain on a
   restricted neighbourhood of size $rw$
3: Secon phase: greedy choices of the reference SNPs. The summarised
   SNPs are searched in the extended neighbourhood of size $w$
4: Optimization of the summarized SNPs
```

3.3 Compression of other variants

The SNPack 1.0 file format cannot handle some kinds of variants such as multiallelic SNPs, multiallelic indels and structure variants (which includes nuclear mitochondrial insertions, duplications, copy number variations). Those variants are not covered by SNPack 1.0 because they need a more complex set of information.
Note that, due to the SNPack's algorithm and file format .pck and due to the .bim file format (which stores the basic information about the variants), the SNPack format can store not only biallelic SNPs, but also biallelic indels. In fact, the .bim file has two columns for the two alleles which are, in case of biallelic SNPs, two letters (one for each allele). In case of indels, those columns can be filled with two strings representing the alleles.
In order to provide a more complete tool we developed a new file format and a new compression algorithm for the compression of all the variant types not covered by SNPack 1.0 (which are all the kinds of variant except biallelic SNPs and biallelic indels).
The variant which are not covered by SNPack are only a small fraction of the total, for this reason the main goal during the development of this new format has been the flexibility.
Note that the SNPack file format, such as other file formats, does not keep the information on the phase, that is, in the case of the heterozygous genotype, the information about which copy of the gene has a particular allele. For this reason, we decided that the phase information would not be kept even for coding the other kinds of variants.
To store the genotype a Huffman encoding [Huffman et al., 1952] is adopted. A prior analysis showed that most variants manifest an allele with a frequency higher than the 90%, for this reason a Huffman encoding results a good choice because it stores the most common genotype configurations using a minor number of bits.
The compression algorithm requires a PLINK .fam file with the information about the individuals (that is the same used to compress the biallelic variants) and a VCF file with the variants to store.
The new file format is composed by three files:

- The PLINK .fam file which describes the individuals.

- A VCF file which contains all the variants information (it has the role of the .bim file for the PLINK format).
  This file is composed by the firsts eight columns of the VCF file required by the algorithm.

- A binary file containing all the genotype data in a compressed format.

The genotype data are stored in the binary file following the order of chromosomes specified in the VCF file. For each chromosome the file contains four bytes for storing the chromosome length (in bytes). Then, for each variant lying in that chromosome two bytes are reserved to store the variant length (in bytes). Information on lengths are useful to browse quickly through the variants and through the chromosomes.
The encoding of all the chromosome variants follows this little header. For each variant the Huffman codebook (i.e. the list of symbols and codewords) is stored, followed by the Huffman encoding of all the individuals.
The codebook of the classic Huffman code is a list containing the symbols, which are the alleles in this case, and their respective codewords. Obviously, the codebook must be stored with the encoded data because it is essential for the decoding of the compressed payload.
The adopted Huffman encoding is the Canonical Huffman code, which has properties that allow to store its codebook in a compact manner i.e. storing only the symbols and their codelength (which is the codeword lenght) is enough to describe the code.
The codebook is composed by: two bytes which indicates the number of symbols that compose the codebook and, for each symbol, two bytes for storing the symbol and two other bytes to store the codelength associated.
Then, the genotype of all the individuals, encoded with the Huffman code, is stored following the individual order specified in the .fam file (detailed information about the encoding of the genotypes and the Canonical Huffman code are described respectively in Subsection 3.3.1 and in Subsection 3.3.2). A scheme of the binary format can be viewed in Figure 3.2.

3.3.1 Encoding the genotype

Encoding those kind of variants with a Huffman code requires to deal with a couple of obstacles. This is a piece of a hypothetical VCF file:

```
22  16051443  .  A  C,G   100  PASS  .  GT  0|0  1|0  0|0  0|0  0|0  0|1
22  16349650  .  G  GT,T  100  PASS  .  GT  0|0  0|1  0|2  0|0  1|2  0|0
```

This piece of file shows two multiallelic variants. The first nine columns contain information about the variants (like chromosome, position and alleles), from the tenth columns is stored, for each individual, the genotype configuration in a vcf cod Recalling that the new file format does not keep track of the genotype phase, the heterozygous genotype configurations with the same alleles should be encoded with the same symbol. In the example above, for

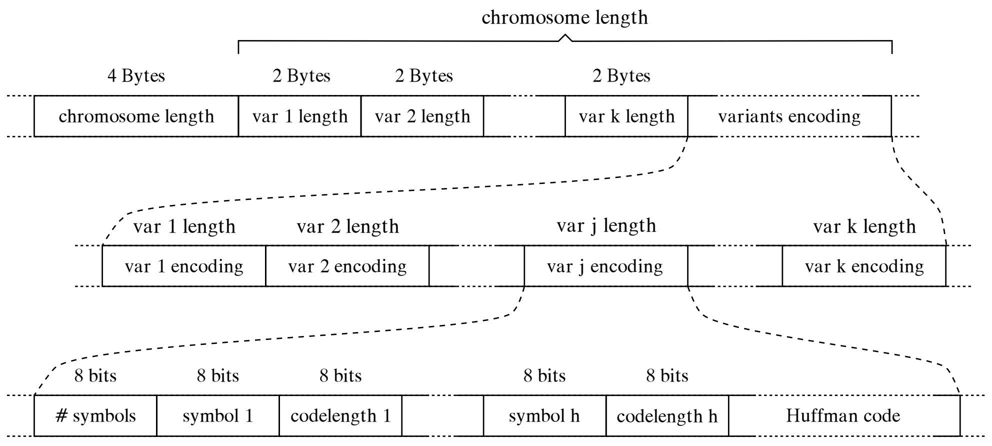
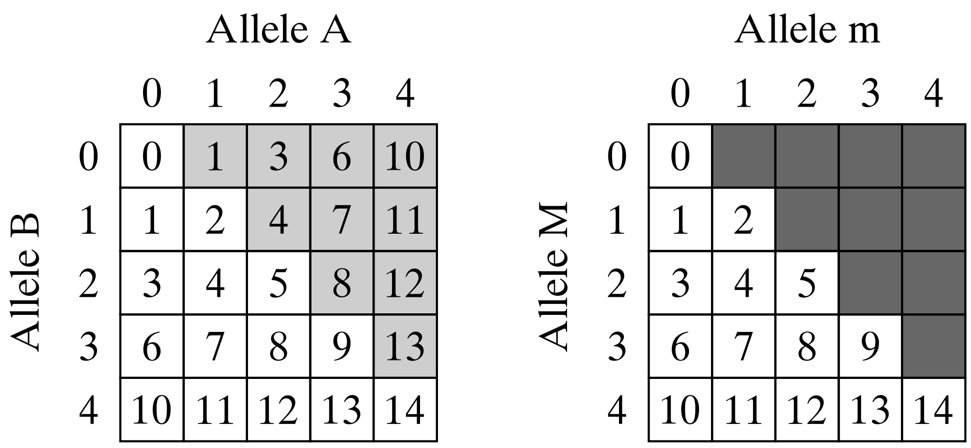


Figure 3.2: A schematic representation of the binary file which stores the genotype of the variants. This figure shows how a chromosome is stored.

the first variant, the second and the sixth individuals are both heterozygous of the alleles A and C and should be encoded with the same symbol although they have different phases. The second variant shows four different genotypes (but theoretically could be more).
The number of genotypes depends on the number of possible alleles. The original version of SNPack works on biallelic variants which lead to three possible genotypes: in that case, thus, for each individual two bits are enough to store its genotype (including the missing genotype configuration). In a multiallelic scenario, on the other hand, the number of bits required depends on the number of alleles. For example, if the number of the variant alleles is three, three homozygous genotype and three heterozygous genotype are possible (plus one missing genotype configuration), this means that the number of bits required to store this variant is 3 (because $\lceil \log_2 7 \rceil = 3$).
We decided to assign a number to each of the possible genotype configurations (this number will be called *internal code*), Figure 3.3a shows in a graphical representation an example of a variant which presents five different alleles. Clearly, due to the fact that the same heterozygous genotypes are coded with the same symbol, the encoding scheme is a symmetric matrix. This means that, as shown in Figure 3.3b, only the triangular lower matrix can be considered, where *Allele m* is the allele with the smaller VCF code and *Allele M* is the allele with the other allele.
Let N be the number of the variant alleles, the number of possible genotypes



(a) Suppose that *Allele A* and *Allele B* are the two alleles which compose the genotype of an individual, since the algorithm does not keep track of the phase the genotypes composed by the same alleles should be represented by the same integer. For this reason the matrix is symmetric.

(b) Since the matrix is symmetric the algorithm calls *Allele m* the allele with the smaller VCF code and *Allele M* the other one, then it considers the lower triangular matrix.

Figure 3.3: Matrix that shows the internal genotype representation. It defines a relationship between an unordered pair of allele and a positive integer value.

is easy to calculate:

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

The conversion from the VCF values to the internal code is simple: let m and M be the smaller and the bigger VCF allele code ($0 \leq m \leq M < N$) respectively, the internal code c is:

$$c = \frac{M(M+1)}{2} + m$$

The conversion in the other direction requires the solution of the equation:

$$c = \frac{x(x+1)}{2}$$

and the VCF allele values are:

$$M = \lfloor x \rfloor = \left\lfloor \frac{-1+\sqrt{1+8c}}{2} \right\rfloor \quad \text{and} \quad m = c - \frac{M(M+1)}{2}$$

This defines a simple one to one relationship between an (unordered) pair of alleles and an integer value. Note that the conversion from the internal code and the pair of alleles requires the computation of a square root which is expensive. For this reason, the implementation of the algorithm uses a precomputed table to decode small c up to 64 (obviously, if c is not present in the table m and M are calculated at runtime).
The number of alleles for indels and structure variations can be indefinitely high but, in practice, it is very hard to find a variant with a number of alleles higher than five or six (that leads to a maximum c value of 21), this means that the adoption of the precomputed table avoids the square root computation for almost all the variants.

3.3.2 The Canonical Huffman code

The Canonical Huffman code is an equivalent variant of the classic Huffman code [Huffman et al., 1952]. Its strength is the compact representation of the codebook, i.e. only the list of symbols and codelengths is enough to describe exhaustively the codebook.
This property is guaranteed by the construction procedure of the code which is composed of few simple steps:

1. Computation of the classic Huffman code.
2. Sort the N codewords firstly by ascending codelength and secondly by alphabetical value (or according another sorting criteria).
   Let s_i, c_i and l_i be respectively the symbol, the codeword and the codelength at position i ($0 \leq i < N$).
3. Replace c_0 with a string of l_i zeroes.
4. Replace c_i with the binary number $c_{i-1} + 1$. If $l_i > l_j$ append $l_i - l_{i-1}$ zeroes at the end of the codeword to reach the codeword lenght l_i.

A Canonical Huffman code construction example is shown in Figure 3.4.
Another property of this code is that if a codeword c_i is longer than a codeword c_j (that is $l_i > l_j$), then the relation between the two binary codes is $c_i > c_j$.

A	11
B	0
C	101
D	100

(a) Example of (non canonical) codebook generated by the Huffman algorithm.

B	0
A	11
C	101
D	100

(b) The codebook is first sorted by the code-length.

B	0
A	10
C	110
D	111

(c) This is the canonical codebook produced following the rules described at the points 3 and 4.

Figure 3.4: Example of a construction of a Canonical Huffman code starting from a classic Huffman code.

Chapter 4

Experimental results

In this chapter will be described the dataset adopted for the tests, what kind of tests has been conducted, how those tests has been conducted and what performance parameters are taken in considerations.
All the tests are applied to the 1000G phase 3 release dataset described in the Section 4.1.
Due to the different nature of the algorithms, the dataset has been split in two: one part includes all the biallelic SNPs and indels, while the other part includes the remaining variants. Some algorithms have been tested only on the biallelic SNPs and indels, while other algorithms have been tested on the entire dataset.
All the tests have been performed on a machine equipped with two quad core Intel Xeon E5450 and 16GB of RAM.

4.1 Dataset

The dataset adopted for the experiments is the 1000G phase 3 release conceived and developed by the 1000 Genome Project.

4.1.1 1000 Genome Project

1000 Genome Project [1000 Genomes Project Consortium, 2012] is one of the major human genome projects and it has the purpose of establishing the most detailed catalogue of human genetic variations.
To do this it was planned to sequence the DNA of at least one thousand of anonymous individuals from different ethnics groups.
The aim of the 1000 Genomes Project is to discover, genotype and provide accurate haplotype information on all forms of human DNA polymorphism in

multiple human populations. Specifically, the goal is to characterize over 95% of variants that are in genomic regions accessible to current high-throughput sequencing technologies and that have allele frequency of 1% or higher in each of five major population groups (populations in or with ancestry from Europe, East Asia, South Asia, West Africa and the Americas). Because functional alleles are often found in coding regions and have reduced allele frequencies, lower frequency alleles (down towards 0.1%) will also be catalogued in such regions.

4.1.2 Dataset characteristics

The dataset on which the tests are done is the 1000G phase 3 release of the 1000 Genome Project, consisting of 84801880 variants typed for a population of 2504 human individuals from 26 populations which can be categorised into five super-populations by continent: East Asia, South Asia, Africa, Europe and America. The dataset composition can be viewed in Table 4.1.
The dataset is provided in the VCF (Variant Call Format) 4.1 file format [Danecek et al., 2011], which is a text-based file format for storing genetic variations (briefly described in the Section 2.1).
One file is provided for each chromosome containing, for each variant present in the chromosome, a set of basic information (among which there are position, alleles, variant ID) and the genotype for each each individual.

Chr	Biallelic variants	Other variants	Total variants
1	6433177	34917	6468094
2	7042490	39110	7081600
3	5799246	33030	5832276
4	5698914	33671	5732585
5	5235101	30662	5265763
6	4994406	29713	5024119
7	4689479	27236	4716715
8	4569575	27530	4597105
9	3539764	20923	3560687
10	3969255	22964	3992219
11	4022210	23418	4045628
12	3846032	22396	3868428
13	2841459	16457	2857916
14	2639628	15439	2655067
15	2410945	13744	2424689
16	2681010	16939	2697949
17	2315827	13461	2329288
18	2254078	13107	2267185
19	1820925	11581	1832506
20	1802645	10196	1812841
21	1098435	7103	1105538
22	1096485	7062	1103547
X	3435430	32663	3468093
Y	61932	110	62042
Total	84298448	503432	84801880

Table 4.1: The dataset composition. The table reports the number of biallelic variants (i.e. biallelic SNPs or indels) and the number of the other kind of variants for each chromosome and for the whole dataset.

4.2 Parameter tuning

The proposed improvements bring new parameters that affect the performance of the compression of biallelic SNPs and indels, i.e. the percentage of the maximum neighbourhood size r and the chance of enabling/disabling the JIT heuristic (used for the calculation of the gain in the first phase of the greedy allocation of the variants).
Those parameters are in addition to the parameters used in the SNPack 1.0 library, which are the maximum extension of the neighbourhood w and the number of chunks in which the chromosomes are split for parallelising the computation.

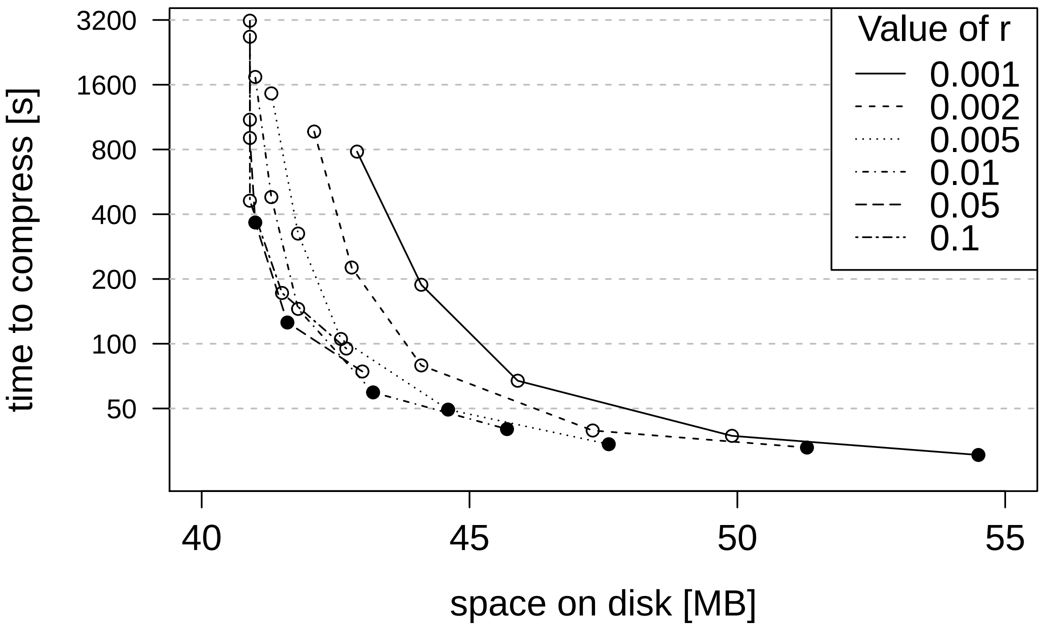


Figure 4.1: Tests performed in order to find the best parameter configuration. Each circle represents a single configuration, each line indicates a set of tests with a fixed r (reported in the legend) and a variable w which goes from 10 kb to 200 kb. The test reported in this figure have the JIT Heuristic disabled.
The black circles indicate the Pareto front identified among those configurations.

Several parameter configurations are tested to tune the parameters. The tests have been performed on chromosome 22, supposing that the obtained results on this chromosome are similar to those obtained if the algorithm was applied to the entire dataset.
The tested parameter configurations are all the combination for r from 0.001 to 0.5, w from 10 kb to 200 kb, with and without the JIT heuristic. Then, we plotted the obtained results on a graph with compression time on the abscissa and the occupied space on the ordinate.
On this graph, we identified the Pareto front (which is the set of solutions that are optimal according to one or both the two criteria, it can be viewed in Figure 4.1) and we selected eight optimal configurations, from the one with the highest compression ratio (but with the highest compression time), to the one with the smallest compression time (but with the smallest compression ratio). Those configurations are wrapped in a new parameter named *compression level* (and are reported in Table 4.2).
The compression of the other variants does not require any parameter, therefore it does not need to be tuned.

compression level	1	2	3	4	5	6	7	8
w[kb]	10	10	10	10	20	20	20	50
r	0.001	0.002	0.005	0.010	0.005	0.010	0.050	0.050
JIT heuristic	yes	yes	no	no	no	no	no	no

Table 4.2: Selected parameter configurations.
Recall w is the maximum neighbourhood window and r is the fraction of w used gain computation first phase.
The higher is the compression level, the higher is the compression ratio, the lower is the compression level, the smaller is the compression time.
In those configuration the chunk number is not present because its effects on the performance depend on the machine parallelism capabilities. The default chunk number setting is 4 times the number of processor cores.

4.3 Tests

The different file formats and tools have different characteristics, such as the capability to compress some kinds of variants or the file type required in input. For this reason, a set of file types has been selected to be the starting point of the tests. These file formats are: the PLINK text-based format (.ped/.map), the PLINK binary format (.bed/.bim/.fam) and the VCF file. The tools used for the tests are:

- **new SNPack version**: it can compress biallelic variants starting from .bed/.bim/.fam files. It can compress all the other variants starting from a VCF file that contains those variants.

- **SNPack 1.0 library**: it can compress biallelic variants starting from .bed/.bim/.fam files.

- **PLINK 1.07**: it can compress biallelic variants. Three tests are performed on the PLINK file formats: the tests involve the binary format, the gzipped version of the binary format and the text-based format.
The PLINK version used in the tests is the current stable version, the 1.07. Since the starting file format of this test is the .bed/.bim/.fam, the PLINK compression time is 0 (but for the last PLINK test the gzip compression time is considered).
The comparison with the PLINK file formats (both the binary form .bed/.bim/.fam and the text-based form .ped/.map) are based on the space on disk required and the loading time.

- **bcftools 1.1**: this is the tool adopted to test the BCF2 file format. The BCF2 file format can compress all kind of variants from the VCF file. The bcftools version adopted is the 1.1.

4.4 Results

This chapter shows the computational results on the tests described previously. For each format is reported the compression time, the space on disk required, and the loading time.

4.4.1 Comparison with SNPack 1.0

The comparison with the previous version of the software shows the performance improvements produced. The change performed to the code affect only the compression algorithm (and not the decompression algorithm), in fact the compression time is drastically reduced and, at the same time, the compression factor is increased. This, however, affects indirectly the loading time, because a smaller file requires less time to be read.
Ah shown in Table 4.3, the new SNPack version is characterized by compression time extremely shorter than the SNPack 1.0. Only one parameter configuration shows a compression time higher than the SNPack 1.0, but it stores the dataset using only two thirds of the disk space.
The new SNPack version has a compression ratio higher than the SNPack 1.0 (even with the fastest parameter configuration which obtains the smallest compression ratio).
The loading time is similar for both versions because the loading algorithm is the same. The new SNPack version has a loading time slightly shorter than the previous version because of the smaller size of the compressed files.
Figure 4.2 shows a plot in which the compression time (in a logarithmic scale) and the space required by the two versions are compared.

Tool	Compr. level	Compr. time (s)	Disk space (MB)	Loading time (s)
SNPack 1.0	-	12867	6730,416	880
SNPack new version	1	2445	6148,312	728
	2	2500	5873,073	703
	3	2632	5564,953	697
	4	2880	5394,967	695
	5	3087	5282,891	679
	6	3560	5174,409	665
	7	7467	5037,106	660
	8	18276	4995,943	662

Table 4.3: Compression performance of the two versions of SNPack.
The space required includes, beside the .pck file, the .fam and the .bim files which are necessary to decompress the data.

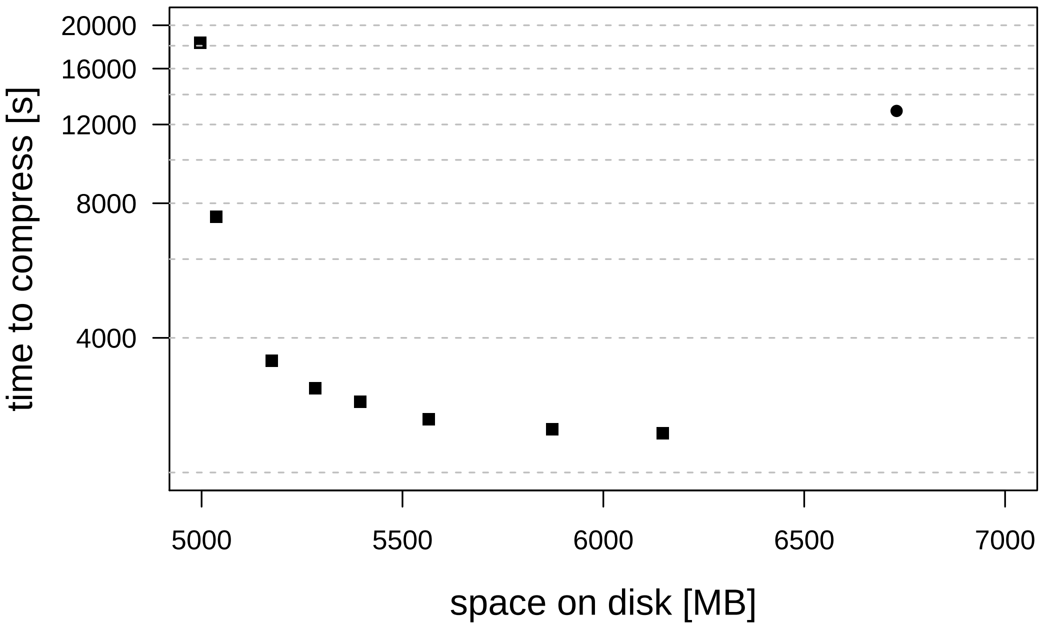


Figure 4.2: Space required (on the X axis) and the compression time (on the Y axis in a logarithmic scale) of both the SNPack 1.0 and the new version. The circle represents the SNPack 1.0 and the squares represent the eight compression levels of the SNPack new version.

4.4.2 Comparison with the other formats

The comparison with the other file formats are made either on the biallelic SNPs and indels dataset or on the entire dataset, depending on the characteristics of the format.

Comparison on the biallelic variants dataset

The comparison on the biallelic variants is conducted on the file formats which are not able to compress the other kind of variants, listed in the Table 4.4. The table reports also the performance of the associated tools.
Since the PLINK's formats are considered as starting points, the compression time related to those format is 0 (the GZIPped version compression time consists on the GZIP compression time).
The new SNPack version is run with a compression level of 5, which is a good trade-off between compression time and compression ratio.
As shown by the table, only the GZIPped version of both the binary PLINK and the new SNPack format have an higher compression ratio than the new SNPack version.
The cons of those configuration is the need of a fully decompression of the GZIP file to access the data, that leads to a high loading time (which, in the case of the GZIPped binary PLINK, reaches and exceeds ten times the new SNPack version loading time) and an additional great amount of space to store the uncompressed files.
The new SNPack version performs extremely better than all the other tools, with respect to the loading time. It is also characterised by excellent performance with respect to both the compression time and the disk space required.

Format	Compr. time (s)	Disk space (MB)	Loading time (s)
PLINK text-based	0	877476,633	237344
PLINK binary	0	54807,274	8481
GZIPped PLINK binary	1495	3707,188	9344
SNPack 1.0	12867	6730,416	880
New SNPack version	3087	5282,891	679
GZIPped new SNPack version	3624	2917,913	805

Table 4.4: Compression performance of the tools working on the biallelic variant dataset.

Format	Compr. time (s)	Disk space (MB)	Loading time (s)
New SNPack version	4532	5717,431	896
Bcftools	26919	11958,774	8894

Table 4.5: Compression performances of the tools working on the whole dataset. The new SNPack version ran with a compression level 5.

Comparison on the entire dataset

This test involves the file formats which can compress even the multiallelic variants. The tools included in this test are listed in the Table 4.5.
The new SNPack version performs extremely better than the other tools for what concerns all the performance parameters took in consideration. The main factor at the basis of this is the well performing algorithm for the biallelic variants which constitute the greater part of the dataset.

Conclusions and future works

The main purpose of this work was to improve the state of the art SNPack 1.0 library, which is an algorithm for the compression and fast retrieval of biallelic variants data.
The improvements produced work in two directions: 1) improvement of the performance of biallelic variants compression algorithm and 2) design and development of a flexible algorithm for the compression and fast retrieval of all other kinds of variants (which are all the multiallelic variants). The improvements on the compression of the biallelic variants are two: the summary optimization and the speed up of the gain computation.
The summary optimization algorithm leads to a consistent gain in terms of compression factor with an almost negligible time cost.
The speed up of the gain computation drastically accelerates the first phase of the compression algorithm and makes unnecessary the adoption of the JIT heuristic unless one wants to trade compression rate for shorter compression time.
All the improvements on the compression of the biallelic data are produced without changing the SNPack 1.0 file format, i.e. from a compressed file is impossible to know if it is built by SNPack 1.0 or by this new version, thus retaining consistency and usability of files compressed with the older version of the library.
The experimental results, obtained on the 1000G phase 3 release dataset, show the outstanding performance of the new SNPack version. This new version presents better compression time and compression ratio than SNPack 1.0 (with a consequently shorter loading time due to the smaller file size).
Furthermore, the comparison with the other formats and tools highlights the superior performance of the new SNPack version for the compression on both the biallelic and the multiallelic variants. If one wants to obtain even higher compression rates, our new format can be further GZIPped, obtaining a smaller file than the corresponding GZIPped binary PLINK file. This consequently leads to the best performances in terms of compression rate but it requires the complete unpacking of the GZIP archive before the access to

the data, this consequently leads to a high loading time and it requires an additional great amount of space to store the uncompressed files.
In the immediate future a tool for the compression and fast retrieval of genetic variants based on the SNPack library will be developed. The reason for this is to provide a tool for easily compressing and accessing genetic data in the SNPack format that can be used directly by the command line interface. Another, perhaps more ambitious, enhancement is the adaptation of the SNPack algorithm to the probabilistic variant data (used for example in IMPUTE [Marchini et al., 2007]) which, for each individual genotype and for each variant, is composed of a probability distributions on the possible variant alleles.

Bibliography

1000 Genomes Project Consortium (2012). An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65.

Allison, L. (2011). *Fundamental Molecular Biology.* CourseSmart Series. Wiley.

Brandon, M. C., Wallace, D. C., and Baldi, P. (2009). Data structures and compression algorithms for genomic sequence data. *Bioinformatics*, 25(14):1731–1738.

Bush, W. S. and Moore, J. H. (2012). Genome-wide association studies. *PLoS computational biology*, 8(12):e1002822.

Christley, S., Lu, Y., Li, C., and Xie, X. (2009). Human genomes as email attachments. *Bioinformatics*, 25(2):274–275.

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., et al. (2011). The variant call format and vcftools. *Bioinformatics*, 27(15):2156–2158.

Grada, A. and Weinbrecht, K. (2013). Next-generation sequencing: methodology and application. *Journal of Investigative Dermatology*, 133(8):e11.

Huffman, D. A. et al. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Li, H. (2011). Tabix: fast retrieval of sequence features from generic tab-delimited files. *Bioinformatics*, 27(5):718–719.

Marchini, J., Howie, B., Myers, S., McVean, G., and Donnelly, P. (2007). A new multipoint method for genome-wide association studies by imputation of genotypes. *Nature genetics*, 39(7):906–913.

Neri, G. and Genuardi, M. (2010). *Genetica umana e medica.* Elsevier.

Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M. A., Bender, D., Maller, J., Sklar, P., De Bakker, P. I., Daly, M. J., et al. (2007). Plink: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575.

Qiao, D., Yip, W.-K., and Lange, C. (2012). Handling the data management needs of high-throughput sequencing data: Speedgene, a compression algorithm for the efficient storage of genetic data. *BMC bioinformatics*, 13(1):100.

Sambo, F., Di Camillo, B., Toffolo, G., and Cobelli, C. (2014). Compression and fast retrieval of SNP data. *Bioinformatics*, 30(21):3078–3085.

Storer, J. A. and Szymanski, T. G. (1982). Data compression via textual substitution. *Journal of the ACM (JACM)*, 29(4):928–951.

Wall, J. D. and Pritchard, J. K. (2003). Haplotype blocks and linkage disequilibrium in the human genome. *Nature Reviews Genetics*, 4(8):587–597.

Wang, C. and Zhang, D. (2011). A novel compression tool for efficient storage of genome resequencing data. *Nucleic acids research*, 39(7):e45–e45.

Watson, J. D., Crick, F. H., et al. (1953). Molecular structure of nucleic acids. *Nature*, 171(4356):737–738.