Advances in Genomic Data Compression

Divon Mordechai Lan

Australian Centre for Ancient DNA
School of Biological Sciences
Faculty of Sciences, Engineering and Technology
University of Adelaide

This thesis is submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

July 2022

Citation listing

The publications included in this thesis:

Chapter 1:

Divon Lan, Raymond Tobler, Yassine Souilmi, Bastien Llamas, genozip: a fast and efficient compression tool for VCF files, *Bioinformatics*, Volume 36, Issue 13, July 2020, Pages 4091–4092, https://doi.org/10.1093/bioinformatics/btaa290

Chapter 2:

Divon Lan, Ray Tobler, Yassine Souilmi, Bastien Llamas, Genozip: a universal extensible genomic data compressor, *Bioinformatics*, Volume 37, Issue 16, 15 August 2021, Pages 2225–2230, https://doi.org/10.1093/bioinformatics/btab102

Chapter 3:

Divon Lan, Glughug Purnomo, Ray Tobler, Yassine Souilmi, Bastien Llamas: Genozip Dual-Coordinate VCF format enables efficient genomic analyses and alleviates liftover limitations (preprint) bioRxiv 2022.07.17.500374, https://doi.org/10.1101/2022.07.17.500374

Appendix (online open access):

Lan, Divon, The Variant Call Format - Dual Coordinate Extension (DVCF) Specification. figshare. 2021, online open access. https://doi.org/10.6084/m9.figshare.14685816.v3

Thesis abstract

The rapid growth in the number of individual whole genome sequences and metagenomic datasets is generating an unprecedented volume of genomic data. This is partly due to the continuous drop in the cost of sequencing as well as growth in the utility of sequencing for research and clinical purposes. We are now reaching a point whereby the lion share of the cost is shifting from the actual sequencing to processing and storing the resulting data.

With genomic datasets reaching the petabyte scale in hospitals and medium to large research groups, it is clear that there is an urgent need to store the data more efficiently - not only to reduce current costs, but also to make sequencing even more affordable to an even larger set of use cases, thereby accelerating the pace of adoption of genomic data for a widening range of research projects and clinical applications.

In Chapter 1 of this thesis, I lay the groundwork for a new approach to compressing genomic data—one that is based on an extensible software platform, which I called Genozip. This initial proof of concept allows compression of data in a widely used format, namely the Variant Call Format, or VCF (Danecek et al. 2011). In Chapter 2, I expand on the work of Chapter 1, showing how the software architecture is designed to support the addition of genomic file formats, compression methods, and codecs. Benchmarking results show that Genozip generally performs better and faster than the leading tools for compression of common genomic data formats such as VCF, SAM (Li et al. 2009) and FASTQ (Cock et al. 2010).

In Chapter 3, I take a detour from compression, and demonstrate how potentially Genozip, with its detailed internal data structures for genomic file processing, could be used for other types of data manipulation. As an example, I introduce DVCF, or Dual-coordinate VCF—an extension of the VCF format that allows representation of genetic variants concurrently in two coordinate systems defined by two different reference genomes (Lan 2021). It is possible to use a DVCF file in a pipeline where each step of the pipeline accesses the data in either of the coordinate systems. I also developed novel methods for lifting over data from one coordinate system to another, and show the superiority of my methods compared to the two leading tools in that space, namely GATK LiftoverVCF (McKenna et al. 2010) and CrossMap (Zhao et al. 2014).

Overall, the Genozip software package is a high quality and versatile bioinformatic tool that is already adopted by dozens of research and clinical laboratories worldwide. Through reduction of the cost of whole genome sequencing data processing and storage, Genozip is likely to further encourage the use of genomics in research and clinical settings.

Thesis declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint award of this degree.

The author acknowledges that copyright of published works contained within the thesis resides with the copyright holder(s) of those works.

I give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

18/07/2022

Acknowledgements

To my supervisors, Bastien Llamas, Yassine Souilmi and Ray Tobler, thank you for taking the risk with a PhD student who is double the age of his peers and with no background in biology. I have learnt a tremendous amount from you - not just in the domains of bioinformatics, population genetics and ancient DNA, but also about how the academic world works (a foreign land to me, coming from industry) and how to go about conducting academic research.

To João Teixeira, both for the initial introduction, as well as the long and fascinating conversations.

To Channé, my wife, my best friend, for following me to Australia, for tolerating the long nights in which I would teleport to an alternative universe of coding, and for keeping me grounded.

Preface

I spent my late 30s and most of my 40s travelling around developing countries in Africa and Asia - 9 years out of those on behalf of Google and after that on my own ventures. At Google, I initially worked as a member of a small team that started Google's operations in Africa, before Africa had any submarine cables connecting it to the Internet. At the time, the only connectivity available was through satellite connections and a 1 Mb/s connection would cost around US$10,000/month. The urge to bring the benefits of the Internet to the people and with it the promise of unrestricted access to information and knowledge was burning in my bones. I had one foot in Google's engineering organisation, working with many of the product and engineering teams in Mountain View, California and other offices around the globe, on product features, adjustments and localisation for developing countries. The other foot was in Africa where I worked with governments, telecom companies, the media and early local technology innovators to spread the gospel of the Internet and facilitate physical and cultural access to its infrastructure. Over time, as Google opened offices and hired local teams in several of the leading countries in Africa with booming economies, I found myself following the frontier of Internet access deeper into Africa's more challenging environments. It is there, in the war-torn tropical hills of the stunningly beautiful Kivu province of the Democratic Republic of Congo, in next door Rwanda, in the arid plains of Hargeisa in Somalia, in the busy lawless markets of Bujumbura, Burundi or the city of Monrovia, Liberia—cautiously recovering from the "blood diamonds" war—that I found the most committed, inspiring individuals which, with a grand vision and minimal material resources, were transforming the capacity of their fellow citizens to access information. This gave me a tremendous sense of purpose, and a reason to continue to do my work. Over time, my role expanded to cover the world's 100 or so least developed countries, and my travel itinerary grew to cover places like Afghanistan, Myanmar, Kyrgyzstan, Bhutan and Papua New Guinea. On one of those trips, to Cambodia, I was giving a presentation at a conference on behalf of Google, when a Cambodian woman sitting in the front row was busy tweeting seemingly everything I was saying. Being media-aware, that made me extremely nervous and unnaturally cagey about my choice of words. Later, the same woman turned up to several of the other meetings I attended during that week—with many of the private sector and government actors in the Internet ecosystem in Cambodia. She was one of those inspiring people and had her hands in everything. Six months later Channé and I were married, and I unexpectedly spent the following 7 years living in Cambodia. We recently celebrated our 10th anniversary.

Over all these years of wandering around the world, and in particular dealing with languages (I was involved in localising Google user interface and its various machine learning algorithms to the languages of “my” countries), I had become increasingly interested in linguistics, anthropology, ethnography, the hard-to-understand issue of ethnic identity and how it ties to language and to perceived ancestry. Later, at age 47, I felt the need for a quantum change in my life. That feeling was now familiar—I had followed it in the past several times, bungee-jumping to an unknown future and never regretting it. I decided to embark on my long-delayed vague plan of pursuing a PhD—and for just a short while, to stop being a jack of all trades and master of none, and immerse myself in pure research without distractions. However, I had no clue *what* I would research? I began canvassing the Internet for ideas—what could I possibly do that ties the only hard-skill that I actually possess, computer science, with my interest in humanities stuff? It was then that I stumbled upon a population genetics project in need of a PhD student, related to a particular region of Papua New Guinea, on some Internet website (its name I have since forgotten), and I reached out to the researchers. Shortly after, I was at my desk in my condominium in Bangkok, on a Skype video call with Ray Tobler and João Teixeira, the authors of the article on the website, from the Australian Centre for Ancient DNA at the University of Adelaide in Australia. They told me about the challenge of deciphering the population history of this little-researched region of Papua New Guinea, and how they are planning to go about it using analysis of DNA samples. Since I knew approximately nothing about population genetics at the time and precisely nothing about this region of Papua New Guinea, this looked like a perfect abyss to bungee-jump into and I decided to apply for the project.

Shortly after landing in Adelaide with my family—Channé and our then-3-year old son Sela—in August of 2019, I began getting up to speed with the new world (for me) of bioinformatics, in parallel to reaching out to old friends in Papua New Guinea to plan a field trip, and starting to establish relationships with collaborators at other research institutions. While processing mountains of genomic data, I grew frustrated with how inefficiently the data is represented, and played around with developing a small compression tool for my own VCF files. Then the Covid-19 pandemic started, and all my travel plans got shelved, for what I anticipated would be 3-6 months. I decided to use this “downtime” to work on my compression tool. Since the tool, which I called Genozip, ended up performing quite nicely, my supervisors encouraged me to publish a paper, which became Chapter 1 of this thesis. The pandemic raged on, and the state of South Australia voluntarily cut itself off from the rest of Australia and the world. The city of Adelaide, while enjoying one of the lowest Covid infection rates in the world, essentially became its own planet with no travel permitted to other large cities in Australia or overseas. While my work on the Papua New Guinea project was progressing very slowly, Genozip was progressing in leaps and bounds. More than once I found myself in a near-24 hours of a hyper-focused programming session that would end with sunrise—something I had not done in almost 20 years and was pleasantly surprised to discover that I could still do. After a 9-month sprint, which resulted in publishing another paper (Chapter 2 in this thesis), I swore to myself and to the world that I was done with coding for the foreseeable future, and went back to playing with my bioinformatics toys on a growing pile of DNA samples from Papua New Guinea sitting in my home directory and staring back at me with reproach. Then I ran into a problem where some of the tools I wanted to use required a specific version of the human reference genome (e.g., GRCh37), while others supported only another (GRCh38). Since at the time my analysis pipeline was already based on Genozip-compressed files, I thought that just having VCF files containing both coordinate systems concurrently would be an elegant solution to this problem, and that I could probably just add it to Genozip in a couple of weeks worth of work. Despite knowing better, I submerged into yet another coding project and when I re-surfaced, it turned out that another half year had zoomed by. Chapter 3 is the result of this lapse in self-discipline of mine. At that point, I stopped and looked back, and realised I had inadvertently created a rather useful tool, consisting of many novel methods, and most importantly, which had been adopted and used by hundreds of researchers and clinical geneticists in 41 countries. I then decided, with support of my supervisors, to move Genozip to be the focus of this PhD project, while postponing the completion of the Papua New Guinea project to a later, postdoctoral, time.

Thesis Introduction

Genomic data: an introduction

Processing genomic and other *omic data is a major component of modern biological and medical research that stands at the cross-section of biology and computer science. This activity has given rise to the relatively new field of Bioinformatics, a term that only started appearing in literature in the late 1980s (Figure 1). Moreover, prior to the advent of Next Generation Sequencing in 2005 (Goodwin *et al.*, 2016), the difficulty and cost of genome sequencing was substantial (Sboner *et al.*, 2011), and as a result the volume of data was relatively modest (Narayanasamy *et al.*, 2020). Researchers typically worked with software tools developed by academic peers, which displayed highly variable levels of quality and performance, and clinicians rarely had the need to directly handle genomic files. Indeed, the interest in Bioinformatics, judging by the frequency of the use of the term in the literature, peaked roughly around the time Next Generation Sequencing became available to the research community (red line in Figure 1).

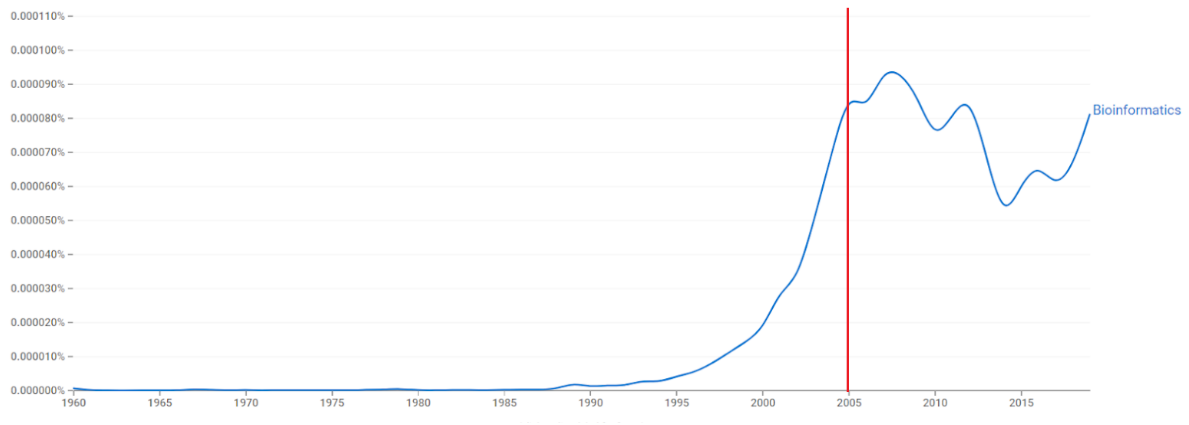


Figure 1: The relative frequency of the word "Bioinformatics" in all books as a % of all words, by year of publication (Source: https://books.google.com/ngrams/graph?content=Bioinformatics&year_start=1960&year_end=2019&corpus=26&smoothing=0). The red line indicates the approximate time Next Generation Sequencing (NGS, aka High Throughput Sequencing) became available to the research community.

In recent years, we have seen a dramatic increase in the quantity of genomic data being generated (Figure 2), driven by the continuous reduction in the price of genomic sequencing that is dropping at a rate faster than Moore's Law (Moore, 1965). Gordon Moore, the founder of Intel, predicted in 1965 that the density of transistors on computer chips, which can be used as a proxy for computation power, would double every two years. This prediction was surprisingly accurate, and continues to hold true until this day.

These price decreases have coincided and even enabled increased usefulness of genomic sequencing in general, and for clinical purposes in particular, driving even more growth in genomic sequencing than expected by improved affordability alone. This increase in volume of genomic sequencing then contributed to economies of scale, further pushing costs down. Moreover, we can expect this trend to further accelerate in the near future—as of today, the typical non-genetics-specialist community doctor does not have the tools to interpret genomic data, in the same way they may do for CT scans or results of a blood test, and as a result current demand for genomic sequencing is mostly driven by genetics specialists (Ha *et al.*, 2018; Nisselle *et al.*, 2021). However, high quality, professional software is now starting to emerge that allows doctors to immediately derive clinically relevant information from genomic sequencing irrespective of their prior genetics expertise. Consequently, we can expect further exponential growth in sequencing in the near future, as a much broader subset of the medical community begins ordering sequence data as a matter of routine practice (Nisselle *et al.*, 2021).
The affordability of sequencing is also a catalyst for the adoption of genomic sequence data in biodiversity analysis, and conservation and evolutionary biology applications, with several large-scale projects such as the Earth BioGenome set to sequence the genomes of the 1.8 million described eukaryotic species (Toward a genome sequence for every animal: Where are we now?; Lewin *et al.*, 2022). Beyond genomics, metagenomic research is booming, the Earth Microbiome Project for example aims to characterise microbial life on Earth (Thompson *et al.*, 2017). Omics has become pervasive across the research landscape, and the announced revolution is truly underway.

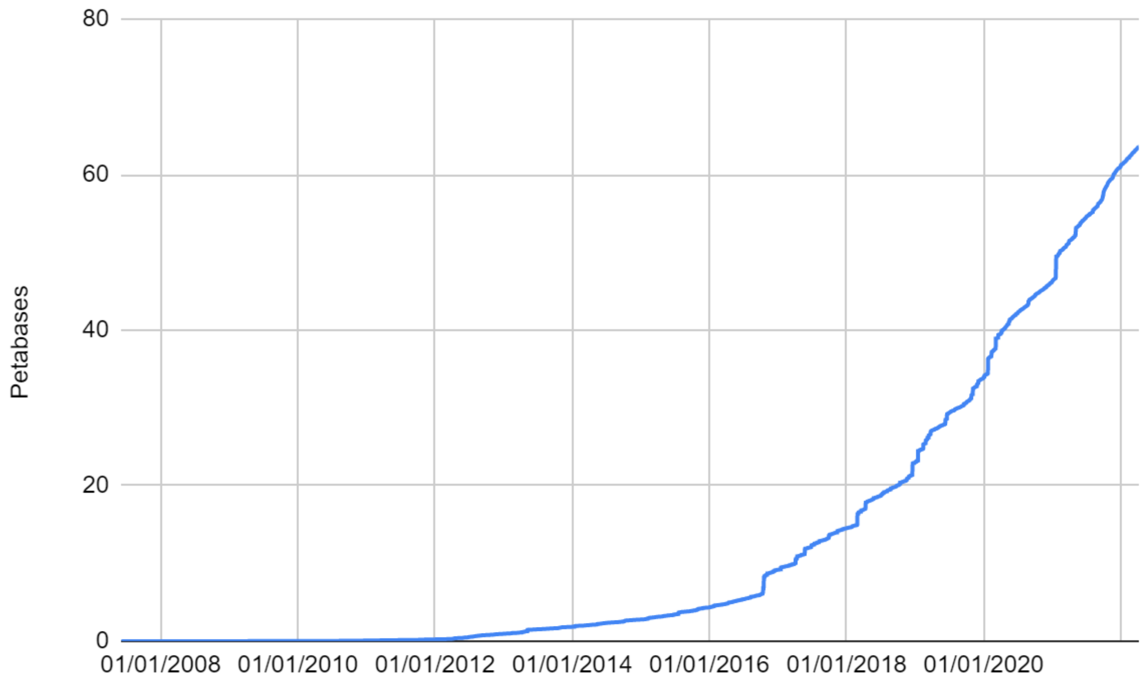


Figure 2: Size of NCBI SRA database between 2008 and 2022 in Petabases (1 Petabase = 10^{15} base pairs) - data downloaded from https://www.ncbi.nlm.nih.gov/sra/docs/sragrowth/

An interesting result of this concurrent drop in price of sequencing and increase in its volume, is the shift of the share of cost attributable to the sequencing itself versus the in-silico processing of the resulting genomic data (Sboner *et al.*, 2011). The Human Genome Project, a project initiated by the Clinton administration in the US that aimed to provide the first fully-assembled human genome, started in 1990. The first incomplete draft was released 13 years later, with the total project cost adding up to US$2.7B (Human Genome Project FAQ). Today, one can sequence a single human individual's complete set of DNA for substantially less than US$1000 in less than 24 hours, though this in fairness is not an apples-to-apples comparison - comparing the total costs involved in an initial de-novo assembly versus just the sequencing costs of a high coverage short-read whole-genome sequencing. Accordingly, the costs associated with genomic data are becoming increasingly dominated by in-silico data processing to convert the raw sequence data, typically received from a sequencing service provider encoded in the FASTQ format, into aligned genomic sequences (e.g., BAM files) and summary files that capture individual genetic features (e.g., genetic variants recorded in VCF files) (Plöthner *et al.*, 2017; Sboner *et al.*, 2011; Schwarze *et al.*, 2020). Even more significant are the costs arising from subsequent data storage that typically extends beyond the lifetime of the project or participant/patient (Krumm and Hoffman, 2020).

Previous genomic data compression methods

Given these cost structure dynamics, the need to compress genomic data has never been more urgent. Unsurprisingly then, there have been a number of attempts to tackle the problem, though most have been piecemeal solutions tailored to specific genomic data types (Table 1).

At present, the most common way to compress genomic files is through use of the gzip format (RFC-1952,1996) - a compression format that will be celebrating its 30th anniversary this year (2022). At the time of its publication in 1992, it represented a major improvement over the previous popular compression tool in the Unix environment, known simply as “compress” that uses the LZW algorithm (Welch, 1984), and as a result quickly gained popularity. The gzip format consists of a small header containing metadata followed by the compressed data itself, and the format allows application-specific custom fields to be added to this metadata. Genomic tools typically use the custom-field capability of gzip to add fields that support indexing of the file, enabling rapid random-access to particular data – an extension of gzip called BGZF. BGZF was made popular by the samtools (Li *et al.*, 2009) and bcftools (Danecek *et al.*, 2021) software, and their associated library htslib (Bonfield *et al.*, 2021).

Gzip’s success over its predecessor, *compress*, was partly due to consuming significantly more CPU and memory than *compress*, which was enabled by the rapid improvements in hardware at the time. While the authors of *compress* were targeting the hardware common at the time of its publication (1984), it appears that by 1992 the gzip authors could allow themselves to be significantly more generous, resulting in much better compression, despite the stated motivation for replacing *compress* being a legal rather than technical one (see https://www.gnu.org/software/gzip/).

Three decades have passed since gzip’s release, and, using Moore’s law as an approximation, hardware has become ~30,000 times more powerful. Accordingly, software developers can leverage this significant increase in modern computing power to devise algorithms able to exploit the availability of giga-bytes of memory and dozens of CPU cores, in return for significantly better compression.

A partial list of the previous attempts to replace gzip for alternate genomic compression formats is provided in Table 1. Notably, every one of these software packages can only operate on a small subset of genomic file formats, and most of them only operate on a single file format. In addition, most projects were written as an academic exercise resulting in a manuscript publication, and did not have the economic structure needed to sustain the software for the long term. I deem a software package to be “Abandoned”, if it has no commit on github since 1 Sep 2021 (appoximately 6 months prior to the time that this text was written). This issue of abandonment is a serious one that goes to the core of the funding structure of the genomics space. I will discuss my plans on how to avoid a similar fate for Genozip in the Discussion section.

Table 1: Partial list of genomic compression tools

Software	**File types**	**Key novelty**	**Ongoing source of funding**[1]	**Status**
CRAM (Bonfield, 2022)	BAM	Intended to be a standard, replacing BAM	Taxpayer funded: European Bioinformatics Institute	Taxpayer funded - viable
Illumina ORA (Hnortonill, 2021)	FASTQ	Unpublished	Enancio - startup company acquired by Illumina in 2020 (Enancio)	Commercial - viable
PetaSuite (Genomic data compression and encryption - PetaSuite, 2018)	FASTQ, BAM	Unpublished	PetaSuite - startup company, raised US$2.4M from investors (PetaGene)	Commercial - viable
GTZ (Xing *et al.*, 2017)	FASTQ, BAM	Compression of SEQ[2]	Genetalks - biotech startup company, raised CN¥270M (approx US$42M) from investors (Genetalks)	Commercial - viable
Spring (Chandak *et al.*, 2019)	FASTQ	Compression of SEQ[2] using De novo assembly	None	Abandoned June 2021
IonCRAM (Shokrof and Abouelhoda, 2020)	BAM (IonExpress only)	Compression of IonExpress signals	None	Abandoned 2020
DeeZ (Hach *et al.*, 2014)	BAM		None	Abandoned 2019
Lfqc (Nicolae *et al.*, 2015)	FASTQ		None	Abandoned 2016
DSRC (Roguski and Deorowicz, 2014)	FASTQ		None	Abandoned 2020

ENANO (Dufort Y Álvarez *et al.*, 2020)	FASTQ	Compression of Nanopore QUAL[3]	None	Abandoned July 2020
RENANO (Dufort Y Álvarez *et al.*, 2021)	FASTQ	Compression of Nanopore SEQ[2]	None	Abandoned June 2021
GTC (Danek and Deorowicz, 2018)	VCF	Compression of GT[4]	None	Abandoned June 2020
GTShark (Deorowicz and Danek, 2019)	VCF	Compression of GT[4] using PBWT	None	Abandoned Jan 2020
VCFShark (Deorowicz *et al.*, 2021)	VCF	Improvements of GTShark	None	Abandoned Feb 2021
FastqCLS (Lee and Song, 2021)	FASTQ	Compression of long-read SEQ[2]	None	Abandoned Dec 2020
NanoSpring (Meng *et al.*, 2021)	FASTQ	Compression of Nanopore SEQ[2]	None	New
CoLoRd (Kokot *et al.*, 2022)	FASTQ	Compression of long-read FASTQ	None	New

[1]Data gathered from public sources. It might not be fully accurate, as by its nature, not all funding information is made public.

[2]SEQ means the nucleotide sequence data in a FASTQ, FASTA or SAM/BAM file.

[3]QUAL means the quality score data in a FASTQ or SAM/BAM file.

[4]GT means the genotype (FORMAT/GT) data in a VCF file.

Compressing genomic data is incredibly complex, because the data itself is very diverse - each file format is composed of several different data types, with the compression of each data type possibly benefitting from its own tailored approaches. Some file formats are rather simple - for example, FASTQ contains just three data types: the description text (sequence name and any metadata), the sequence data, and the quality score data. In contrast, BAM and VCF files may contain tens of different fields each ideally requiring its own methods of compression. Moreover, new types of fields are added all the time, as new tools are developed for specific types of analysis, emitting BAM and VCF files with their own proprietary fields - all needing compression.

It is no wonder then, that most of the academic tools listed in Table 1 focus on introducing a new method for one or two fields found within one or two file formats. Genozip, in contrast, is built on an extensible architecture devised to accommodate compression of a large number of types of genomic data. However, for each data type, specific methods are still required. Genozip not only contains many novel methods covering the various data types (see Chapters 1 and 2), but also takes advantage of state-of-the-art techniques already available. For example, recent versions of Genozip use a novel implementation of PBWT (Durbin, 2014) for compressing VCF FORMAT/GT data that is inspired by GTShark (Deorowicz and Danek, 2019), though the PBWT-derived algorithm in Genozip has several important differences. Another example is an algorithm for compressing long-read quality scores inspired by ENANO (Dufort Y Álvarez *et al.*, 2020), which has been modified in Genozip to compress both Oxford Nanopore Technology (aka ONT) and Pacific Bioscience (aka PacBio) data, with further modifications extending its utility to both FASTQ and BAM data (the original ENANO only works on Oxford Nanopore FASTQ files). Other methods, notably the methods in SPRING and NanoSpring that use approximate de-novo assembly for reference-free compression of FASTQ sequence data from short-reads and Nanopore long reads respectively, are notable for their originality and excellence, but have not been adopted in Genozip because they have fundamental differences with Genozip's internal data processing pipeline.

Genozip - a different approach

With Genozip, I set out to explore the following hypothesis:

methods tailored to the structure of genomic data will improve compression rates.

The approach I have taken in Genozip is radically different from both general-purpose compressors and from specialised genomic compressors. In the following, I outline several key features and innovations that distinguish Genozip from alternate genomic compressors that are currently available.

First, owing to my background in the software industry, Genozip was designed as a real product with which users can entrust their precious data - i.e., designed and built with robust software engineering and quality assurance, to ensure quality and maintainability over the years and decades to come. This stands in stark contrast to the majority of specialised genomic compressors that were built at a proof-of-concept level aiming for an academic publication, only to be abandoned shortly after (Table 1, Matthews 2022).

Second, Genozip is founded upon an extensible framework that allows easy evolution along three axes: 1) additional support for more genomic file formats, 2) the addition of “special algorithms” for specific fields within file formats, and 3) the addition of codecs used for final compression of the data from each field, after they are processed by the appropriate field-specific algorithms. Indeed, following Genozip’s initial release in 2020 as a VCF compressor, it has acquired the ability to compress nine further genomic file formats (i.e. SAM/BAM/CRAM, FASTQ, FASTA, GFF3/GVF, 23andMe, PHYLIP, Chain files, Kraken, Illumina locs) as well as generic (i.e., not necessarily genomic) files. The number of codecs employed has grown from 2 to 15 - some of them derived from 3rd party libraries, and some originally developed for Genozip. Within each file format, Genozip is improving with each release, as new special algorithms are added and existing ones improved, to handle the plethora of field types being generated by the rapidly expanding omics disciplines. BAM and VCF files, for example, are generated by many different bioinformatics tools, each generating specialised fields to fit their analysis objective. Genozip has special algorithms to compress fields generated by many popular bioinformatics tools. At the time of writing, Genozip (version 13.0.13) has 49 special algorithms. In addition, constantly evolving sequencing technologies as well as specialised library preparation protocols (for example, treatment with Bisulfite for methylation detection (Frommer *et al.*, 1992)), result in nucleotide and quality score sequences with different statistical properties (Guo *et al.*, 2013; Farrell *et al.*, 2021) that Genozip handles with algorithms and codecs tailored for each case.

Third, any genomics compressor faces a tradeoff between the depth of compression and the speed of random access to a subset of the data within the compressed file, should the compressed file be used in a bioinformatics pipeline. CRAM for example, which aims to replace BAM as the de-facto standard short read alignment format in bioinformatics pipelines (see: https://www.ga4gh.org/cram/), offers very fast access to subsets of data, enabled by making certain design decisions. For example, both CRAM and Genozip divide the source file into blocks and compress each block separately (called *vblocks* in Genozip and *containers* in CRAM) but CRAM’s blocks are relatively small and it does little in terms of exploitation of correlation between fields in the file - two design decisions that inevitably limit compression (Bonfield, 2022). Genozip, on the other hand, optimises for compression, even at the expense of slower random access to subsets of data. Despite this, there are some cases where data subsetting is actually faster in Genozip than CRAM, for example when accessing summary statistics (the --count option in both genozip and samtools).

Fourth, Genozip guarantees losslessness (unless specified otherwise by the user): i.e., compressing a file, and then decompressing it, results in an identical file, byte-by-byte, as verified by MD5 (Rivest, 1992) or Adler32 (Deutsch and Gailly, 1996). This means that all fields need to be reconstructed precisely at the byte level - it is not sufficient that they contain semantically similar information. Table 4 lists some examples in which CRAM produces semantically similar information, but yet is not lossless, while Genozip is. As one can appreciate from these examples, while the lack of lossnessness in CRAM is not significant for pipeline analysis purposes, it is critical for any integrity-verification process that might be used in a long term archival system - a key use case for Genozip. Similar to CRAM, Genozip offers an option of lossy compression in which some fields are modified in ways that usually do not have a significant impact on downstream analysis yet improve compression significantly (see Chapter 1, section 2). However, judging from my direct interaction with Genozip users, almost all of them refrain from using the lossy option and very much value the MD5-verifiable lossless capability of Genozip.

Table 4 - examples of losslessness violations in CRAM

Field	**Issue**	**Source file data**	**CRAM reconstruction**	**Genozip reconstruction**
Any XX:f or XX:B:f field	Floating point - representation	`SAM: 0.100`	`0.1`	`0.100`
Any XX:f or XX:B:f field	Floating point - precision	`SAM: 0.1000000000000000000001`	`0.1`	`0.1000000000000000000001`
QUAL	Representation of “missing quality” in BAM files produced by the Pysam library used by most bioinformatics tools written in Python[1]	`BAM: 0xff000000`	`0xffffffff`	`0xff000000`
SEQ	Sequence case actg->ACTG	`SAM: acctgt`	`ACCTGT`	`acctgt`

[1] In march 2022, following my advice, Pysam was fixed to avoid this issue in files generated going forward: https://github.com/pysam-developers/pysam/issues/1089

Fifth, Genozip takes full advantage of IT environments typical in modern bioinformatics analysis settings: owing to its sophisticated parallelisation algorithms that reduce thread synchronisation points, Genozip can fully utilise tens of CPU cores to accelerate computation. It also takes full advantage of the low seek times of modern solid state drives

(SSD; (El Maghraoui *et al.*, 2010)) to access a Genozip compressed file, in a highly non-linear fashion in some cases, resulting in better compression algorithms that would be significantly more difficult to implement if a Genozip sought linear or near-linear access to the file that is required for optimising seek time in spinning disks.

Sixth, DNA data, in particular human DNA data, may have strict privacy requirements. As the public awareness of the risks of mishandling of DNA data is growing, so too are the ethical and regulatory requirements incumbent upon clinical and research labs ((Rahimzadeh *et al.*, 2016; Office for Civil Rights (OCR), 2021)). It is my anecdotal impression based on my interactions with research and clinical labs around Genozip, that sadly, security and privacy are still implemented as an afterthought in many contemporary bioinformatics projects, and the level of protection of patients' DNA data is questionable. In Genozip, privacy is built-in with powerful industry-standard encryption being enabled with a simple `--password` command line option. In addition, cryptographic MD5 is used to sign files, protecting them both from a modification caused by technical glitch, as well as from malicious tampering.

Seventh, while most bioinformatic analyses are typically conducted in Linux environments, it is also common for bioinformaticians to develop pipelines on their Mac or Windows personal computers prior to deploying to a Linux environment, and it is common for students to use their personal computers for training. These environments, and Windows in particular, are too often neglected by bioinformatics tools. In Genozip, all three operating systems are first-class citizens. While Mac and Linux are fairly similar in most aspects relevant to Genozip owing to their shared Unix ancestry, the Microsoft Windows operating system has a fundamentally different architecture (of which we are concerned mostly with differences in process management, thread synchronisation, and the NTFS file system), which required significant effort to support. At the time of writing, 82% of Genozip installations are on Linux machines, 11% on Mac and 7% on Windows.

Overview of thesis Chapters 1, 2 and 3

The three chapters in this thesis each outline central operational aspects of the Genozip platform, and are presented in chronological order in accordance with the ongoing development of the platform.

Chapter 1 is a paper titled "genozip: a fast and efficient compression tool for VCF files" that was published in Bioinformatics in July 2020 (Lan *et al.*, 2020). It reports the first iteration of

Genozip, as a compressor of VCF files, and sets the foundations for the Genozip architecture. It includes benchmarks against other VCF compressors, and demonstrates the merits of Genozip's approach.

Chapter 2 is a paper titled "Genozip - A Universal Extensible Genomic Data Compressor" that was published in Bioinformatics in February 2021 (Lan *et al.*, 2021). It describes Genozip's advancement to becoming a fully-fledged extensible genomic compressor for multiple types of genomic file formats. It describes the architecture enabling Genozip's extensibility as well as some of the novel algorithms devised, such as Genozip's aligner algorithm that operates at the bit level rather than the traditional k-mer level, allowing very fast compression of FASTQ and unaligned BAM files based on approximate alignment.

Chapter 3 consists of two related but separate bodies of work. First, I developed an extension of the VCF file format, which I called Dual-coordinate VCF (or DVCF), designed to accommodate descriptions of genetic variants in two genomic coordinate systems within a single file (for example, human reference genome versions GRCh37 and GRCh38). The key property of the new format is the concept of dual-renditions: each DVCF file can be *rendered* in either of its two coordinate systems, with each rendition being a VCF file adhering to the VCF specification describing the variants in one of the coordinates, while also retaining the information pertaining to the other coordinate system, thereby maintaining information equivalence between the two renditions and lossless back-and-forth convertibility between them. As I intend to recommend this document as a basis for a possible extension of the VCF, I was not interested in assigning any IP rights in it to a journal, and instead deposited it in the open-access repository Figshare (Lan, 2021). It appears in Appendix 1 of this dissertation.

The second, related, body of work is a manuscript that has been submitted to a journal and deposited on bioRxiv. It is the first attempt to extend Genozip into the analysis space - implementing DVCF in Genozip, together with novel algorithms for lifting over variants from one coordinate system to another. I benchmarked the liftover in Genozip versus the two most widely used tools in the space, namely CrossMap (Zhao *et al.*, 2014) and GATK LiftoverVcf (Broad Institute, 2016), and demonstrate that Genozip is significantly more accurate and also solves certain biases introduced by the liftover process in the other tools.

References

Bonfield,J.K. (2022) CRAM 3.1: Advances in the CRAM File Format. *Bioinformatics*.

Bonfield,J.K. *et al.* (2021) HTSlib: C library for reading/writing high-throughput sequencing data. *GigaScience*, **10**.

Broad Institute (2016) Picard tools.

Chandak,S. *et al.* (2019) SPRING: a next-generation compressor for FASTQ data. *Bioinformatics*, **35**, 2674–2676.

Danecek,P. *et al.* (2021) Twelve years of SAMtools and BCFtools. *Gigascience*, **10**.

Danek,A. and Deorowicz,S. (2018) GTC: how to maintain huge genotype collections in a compressed form. *Bioinformatics*, **34**, 1834–1840.

Deorowicz,S. *et al.* (2021) VCFShark: how to squeeze a VCF file. *Bioinformatics*.

Deorowicz,S. and Danek,A. (2019) GTShark: genotype compression in large projects. *Bioinformatics*, **35**, 4791–4793.

Deutsch,P. and Gailly,J.-L. (1996) Zlib compressed data format specification version 3.3 RFC 1950, May.

DNA sequencing costs: Data *Genome.gov*.

Dufort Y Álvarez,G. *et al.* (2020) ENANO: Encoder for NANOpore FASTQ files. *Bioinformatics*, **36**, 4506–4507.

Dufort Y Álvarez,G. *et al.* (2021) RENANO: a REference-based compressor for NANOpore FASTQ files. *Bioinformatics*.

Durbin,R. (2014) Efficient haplotype matching and storage using the positional Burrows–Wheeler transform (PBWT). *Bioinformatics*, **30**, 1266–1272.

El Maghraoui,K. *et al.* (2010) Modeling and simulating flash based solid-state disks for operating systems. In, *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, WOSP/SIPEW '10. Association for Computing Machinery, New York, NY, USA, pp. 15–26.

Enancio *Crunchbase*.

Farrell,C. *et al.* (2021) BiSulfite Bolt: A bisulfite sequencing analysis platform. *Gigascience*, **10**.

Frommer,M. *et al.* (1992) A genomic sequencing protocol that yields a positive display of 5-methylcytosine residues in individual DNA strands. *Proc. Natl. Acad. Sci. U. S. A.*, **89**, 1827–1831.

Genetalks *Crunchbase*.

Genomic data compression and encryption - PetaSuite (2018) *PetaGene*.

Goodwin,S. *et al.* (2016) Coming of age: ten years of next-generation sequencing technologies. *Nat. Rev. Genet.*, **17**, 333–351.

Guo,W. *et al.* (2013) BS-Seeker2: a versatile aligning pipeline for bisulfite sequencing data. *BMC Genomics*, **14**, 774.

Hach,F. *et al.* (2014) DeeZ: reference-based compression by local assembly. *Nat. Methods*, **11**, 1082–1084.

Ha,V.T.D. *et al.* (2018) Adopting clinical genomics: a systematic review of genomic literacy among physicians in cancer care. *BMC Med. Genomics*, **11**, 18.

Hnortonill,V.A.P. (2021) Introducing DRAGEN original read archive (ORA) - Illumina® informatics blog. *Illumina® Informatics Blog*.

Human Genome Project FAQ *Genome.gov*.

Kokot,M. *et al.* (2022) CoLoRd: compressing long reads. *Nat. Methods*, **19**, 441–444.

Krumm,N. and Hoffman,N. (2020) Practical estimation of cloud storage costs for clinical genomic data. *Pract Lab Med*, **21**, e00168.

Lan,D. *et al.* (2020) genozip: a fast and efficient compression tool for VCF files. *Bioinformatics*, **36**, 4091–4092.

Lan,D. *et al.* (2021) Genozip - A Universal Extensible Genomic Data Compressor. *Bioinformatics*.

Lan,D. (2021) The variant call format - Dual Coordinates extension (DVCF) specification.

Lee,D. and Song,G. (2021) FastqCLS: a FASTQ Compressor for Long-read Sequencing via read reordering using a novel scoring model. *Bioinformatics*.

Lewin,H.A. *et al.* (2022) The Earth BioGenome Project 2020: Starting the clock. *Proc. Natl. Acad. Sci. U. S. A.*, **119**.

Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Matthews, D. (2022) Ex-Google chief's venture aims to save neglected science software. *Nature*, **607**, 410-411

Meng,Q. *et al.* (2021) NanoSpring: reference-free lossless compression of nanopore sequencing reads using an approximate assembly approach. *bioRxiv*, 2021.06.09.447198.

Moore,G. (1965) Moore's law. *Electronics Magazine*, **38**, 114.

Narayanasamy,S. *et al.* (2020) Genomic Sequencing Capacity, Data Retention, and Personal Access to Raw Data in Europe. *Front. Genet.*, **11**, 303.

Nicolae,M. *et al.* (2015) LFQC: a lossless compression algorithm for FASTQ files. *Bioinformatics*, **31**, 3276–3281.

Nisselle,A. *et al.* (2021) Measuring physician practice, preparedness and preferences for genomic medicine: a national survey. *BMJ Open*, **11**, e044408.

Office for Civil Rights (OCR) (2021) HIPAA. *HHS.gov*.

PetaGene *Crunchbase*.

Plöthner,M. *et al.* (2017) Cost analysis of whole genome sequencing in German clinical practice. *Eur. J. Health Econ.*, **18**, 623–633.

Rahimzadeh,V. *et al.* (2016) An International Framework for Data Sharing: Moving Forward with the Global Alliance for Genomics and Health. *Biopreserv. Biobank.*, **14**, 256–259.

RFC 1952 - GZIP file format specification version 4.3

Rivest,R. (1992) RFC1321: The MD5 Message-Digest Algorithm RFC Editor, USA.

Roguski,Ł. and Deorowicz,S. (2014) DSRC 2—Industry-oriented compression of FASTQ files. *Bioinformatics*, **30**, 2213–2215.

Sboner,A. *et al.* (2011) The real cost of sequencing: higher than you think! *Genome Biol.*, **12**, 125.

Schwarze,K. *et al.* (2020) The complete costs of genome sequencing: a microcosting study in cancer and rare diseases from a single center in the United Kingdom. *Genet. Med.*, **22**, 85–94.

Shokrof,M. and Abouelhoda,M. (2020) IonCRAM: a reference-based compression tool for ion torrent sequence files. *BMC Bioinformatics*, **21**, 397.

Thompson,L.R. *et al.* (2017) A communal catalogue reveals Earth's multiscale microbial diversity. *Nature*, **551**, 457–463.

Toward a genome sequence for every animal: Where are we now? *PNAS*.

Welch (1984) A Technique for High-Performance Data Compression. **17**, 8–19.

Xing,Y. *et al.* (2017) GTZ: a fast compression and cloud transmission tool optimized for FASTQ files. *BMC Bioinformatics*, **18**, 549.

Zhao,H. *et al.* (2014) CrossMap: a versatile tool for coordinate conversion between genome assemblies. *Bioinformatics*, **30**, 1006–1007.

Chapter 1

genozip: a fast and efficient compression tool for VCF files

Divon Lan[1]*, Ray Tobler[1], Yassine Souilmi[1†], Bastien Llamas[1†]*

[1] School of Biological Sciences, The Environment Institute, Faculty of Sciences, The University of Adelaide, Adelaide SA 5005, Australia

[†] Equal contribution

* Corresponding authors: DL (divon.lan@adelaide.edu.au) and BL (bastien.llamas@adelaide.edu.au)

Citation: Divon Lan, Raymond Tobler, Yassine Souilmi, Bastien Llamas, genozip: a fast and efficient compression tool for VCF files, *Bioinformatics*, Volume 36, Issue 13, July 2020, Pages 4091–4092, https://doi.org/10.1093/bioinformatics/btaa290

Article history:
Received: 15 January 2020
Revision received: 01 April 2020
Accepted: 27 April 2020
Published: 14 May 2020

Statement of Authorship

Title of Paper	genozip: a fast and efficient compression tool for VCF files
Publication Status	[✓ Published] Accepted for Publication Submitted for Publication Unpublished and Unsubmitted work written in manuscript style
Publication Details	*Bioinformatics*, Volume 36, Issue 13, July 2020, Pages 4091–4092, https://doi.org/10.1093/bioinformatics/btaa290 **Received:** 15 January 2020 **Revision received:** 01 April 2020 **Accepted:** 27 April 2020 **Published:** 14 May 2020 4 citations as of 1/Feb/2022

Principal Author

Name of Principal Author (Candidate)	Divon Mordechai Lan		
Contribution to the Paper	Discovered the methods, developed the software, and wrote the initial version of the paper.		
Overall percentage (%)	90%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	20/04/2022

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Bastien Llamas		
Contribution to the Paper	Supervision, results interpretation and minor edits		
Signature		Date	20/04/2022

Name of Co-Author	Yassine Souilmi		
Contribution to the Paper	Supervision, results interpretation and minor edits		
Signature		Date	20/04/2022

Name of Co-Author	Raymond Tobler		
Contribution to the Paper	Supervision, results interpretation and minor edits		
Signature		Date	20/04/2022

Abstract

Summary

`genozip` is a new lossless compression tool for VCF (Variant Call Format) files. By applying field-specific algorithms and fully utilising the available computational hardware, `genozip` achieves the highest compression ratios amongst existing lossless compression tools known to the authors, at speeds comparable with the fastest multi-threaded compressors.

Availability and implementation

`genozip` is freely available to non-commercial users. It can be installed via conda-forge, Docker Hub, or downloaded from github.com/divonlan/genozip.

1. Introduction

Large genomic projects are becoming increasingly common, resulting in VCF (Variant Call Format; (Danecek *et al.*, 2011)) files comprising thousands of individual genomic datasets. Even in their compressed form, such files are very large (typically several GB), rapidly driving up the cost of long-term data storage and file transfer, and spurring the development of more efficient compression algorithms.

While a handful of new compression algorithms have recently emerged that work by compressing genotypes within VCF files; e.g. (Durbin, 2014; Deorowicz and Danek, 2019; Kelleher *et al.*, 2019), genotypes are only one data type represented in a VCF file, and are often only a minor contributor to the total data content. For example, in the file used as the real-world example in (Durbin, 2014) – File1 in our benchmarks – the genotypes represent only 7.1% of the uncompressed VCF file data. Thus, it is clear that just compressing the genotypes is not sufficient as a compression strategy for VCF files.

Here, we present `genozip`, a lossless compression tool that greatly improves genomic data compression by utilising algorithms specific to the data types common to VCF files. `genozip` can handle VCF files of any ploidy, phasing structure, or variant type with up to 99 alternate alleles per variant, along with any FORMAT and INFO data. While the primary objective of `genozip` is optimal packaging of genomic data for efficient and secure storage and distribution, it also includes capabilities for pipeline analyses.

2. Software description

The `genozip` package runs on all popular operating systems and includes four command line tools – `genozip, genounzip, genocat` and `genols`. `genozip` receives one or more .vcf, .vcf.gz, .vcf.bz2, .vcf.xz, or .bcf files or urls (FTP or HTTP) as input, and outputs one or more .vcf.genozip files, while `genounzip` decompresses .vcf.genozip files back to .vcf or .vcf.gz format and `genols` provides statistics regarding the contents of .genozip files.

For supporting seamless integration into analytical pipelines, `genocat` is provided for accessing data within .vcf.genozip files, and includes options like `--regions` and `--samples` that allow random access to data. Indexing is done as part of the compression and there is no separate indexing step or index file. In addition, the toolset is designed to enable use of standard input/output streams.

For supporting efficient and secure distribution of genomic files that complies with stringent privacy requirements, `genozip` offers encryption of the data with `--password` (using 256 bit AES), including an MD5 signature to ensure data integrity with `--md5`, and the ability to concatenate VCF files with identical samples with `--output` and later split concatenated files back to their components with `genounzip --split`.

We have included several additional options that allow the user to optimise compression for their needs. First, the `--optimize` option improves compression by modifying data in some INFO and FORMAT subfields that do not ultimately impact analytical results — by rounding floating point numbers to 2 significant digits and capping Phred values. Note that in this case the VCF data are modified, and therefore the compression is not lossless. Second, The `--gtshark` option makes use of `GTShark` (Deorowicz and Danek, 2019) as described in the Supplementary Material, resulting in compression ratios that are better than either `genozip` or `GTShark` alone. Finally, the `--vblock` and `--sblock` options allow the user to control the tradeoff between compression and speed of subsetting regions and samples.

Note that some options require the appropriate tools to be installed: compressing .bcf files into .genozip format requires `bcftools`, compressing .xz files requires `xz` (Collin, 2011), decompressing into .vcf.gz requires `bgzip`, using `--gtshark` requires `GTShark` and compressing from a URL requires `cURL` (Hostetter *et al.*, 1997).

3. Benchmark

To evaluate `genozip`'s performance, we compared its compression ratios and speeds on two different VCF files from the 1000 Genome Project (1000 Genomes Project Consortium *et al.*, 2012) – 'File1' and 'File2' (see Supplementary Material) – against a wide range of tools**.** All benchmarks were conducted on the same machine that has 56 physical cores (4 X Intel® Xeon® Gold 6132 CPU @ 2.60GHz) and 755GB of usable memory. More details, including benchmarks against genotype compression tools including BGT (Li, 2016) and GTC (Danek and Deorowicz, 2018) that are not capable of losslessly compressing arbitrary VCF files are available in the Supplementary Material.

For both tested VCF files, File1 which is rich in FORMAT subfields and File2 that is rich in genotype data (see Supplementary Table S1), the compression ratios achieved by `genozip` are considerably higher than other tested tools (Figure 1a). Further, `genozip` offers one of the fastest compression/decompression speeds amongst the tested tools (Figure 1b), indicating that performance gains are achieved without negatively impacting run times. To achieve high processing speeds, `genozip` implements an advanced memory and thread management strategy that scales across 10s of cores (Figure 1c).

4. Conclusion

`genozip` is a user friendly and fully featured compression software that readily integrates into any standard bioinformatics pipeline. `genozip` achieves compression ratios significantly better than other standard tools, by exploiting redundancies in the data that are specific to biological data and that are not evident by textual analysis alone. Moreover, `genozip` achieves significant gains to compression speed relative to other tools by taking full advantage of modern computational hardware, including multi-core processors and multi-gigabyte RAM, whenever available. By default, `genozip` dynamically balances its internal execution pipelines to maximize utilization of all the available resources.

Funding

DL is supported by a scholarship from the University of Adelaide. YS is supported by the Australian Research Council (ARC DP190103705). RT is an ARC DECRA fellow (DE190101069). BL is an ARC Future Fellow (FT170100448).

Conflict of Interest: DL intends to receive royalties from commercial users of genozip.

ACKNOWLEDGMENTS
The authors thank Christian Huber, Heng Li and two anonymous reviewers for comments on the manuscript.

References

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

Adler,M. (2014) Pigz: Parallel Gzip.

Bonfield,J.K. (2014) The Scramble conversion tool. *Bioinformatics*, **30**, 2818–2819.

Chandak,S. *et al.* (2019) SPRING: a next-generation compressor for FASTQ data. *Bioinformatics*, **35**, 2674–2676.

Collin,L. (2011) The xz utils software package. *XZ Utils*.

Danecek,P. *et al.* (2011) The variant call format and VCFtools. *Bioinformatics*, **27**, 2156–2158.

Danek,A. and Deorowicz,S. (2018) GTC: how to maintain huge genotype collections in a compressed form. *Bioinformatics*, **34**, 1834–1840.

Deorowicz,S. and Danek,A. (2019) GTShark: genotype compression in large projects. *Bioinformatics*, **35**, 4791–4793.

Deutsch,P. (1996) DEFLATE Compressed Data Format Specification version 1.3 RFC Editor.

DNA sequencing costs: Data *Genome.gov*.

Durbin,R. (2014) Efficient haplotype matching and storage using the positional Burrows–Wheeler transform (PBWT). *Bioinformatics*, **30**, 1266–1272.

Fips,P. (2009) 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, US Department of Commerce, November 2001. *Link in: http://csrc. nist. gov/publications/fips/fips197/fips-197. pdf*.

Gailly,J.-L. and Adler,M. (2010) GNU gzip.

Goodwin,S. *et al.* (2016) Coming of age: ten years of next-generation sequencing technologies. *Nat. Rev. Genet.*, **17**, 333–351.

Hail Team hail Github.

Hostetter,M. *et al.* (1997) Curl: a gentle slope language for the Web. *World Wide Web J. Biol.*, **2**, 121–134.

Kelleher,J. *et al.* (2019) Inferring whole-genome histories in large population datasets. *Nat. Genet.*, **51**, 1330–1338.

Lan,D. *et al.* (2020) genozip: a fast and efficient compression tool for VCF files. *Bioinformatics*, **36**, 4091–4092.

Lan,D. *et al.* (2021) Genozip - A Universal Extensible Genomic Data Compressor. *Bioinformatics*.

Li,H. (2011a) A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, **27**, 2987–2993.

Li,H. (2016) BGT: efficient and flexible genotype query across many samples. *Bioinformatics*, **32**, 590–592.

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

Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Meyering,J. gzip-1.10 released [stable]. *gzip-1.10 released [stable]*.

Pavlov,I. (2007) Lzma sdk (software development kit).

Rivest,R. (1992) RFC1321: The MD5 Message-Digest Algorithm RFC Editor, USA.

Sadedin,S.P. and Oshlack,A. (2019) Bazam: a rapid method for read extraction and realignment of high-throughput sequencing data. *Genome Biol.*, **20**, 78.

Seward,J. (1996) bzip2 and libbzip2. *avaliable at http://www. bzip. org*.

Sudmant,P.H. *et al.* (2015) An integrated map of structural variation in 2,504 human genomes. *Nature*, **526**, 75–81.

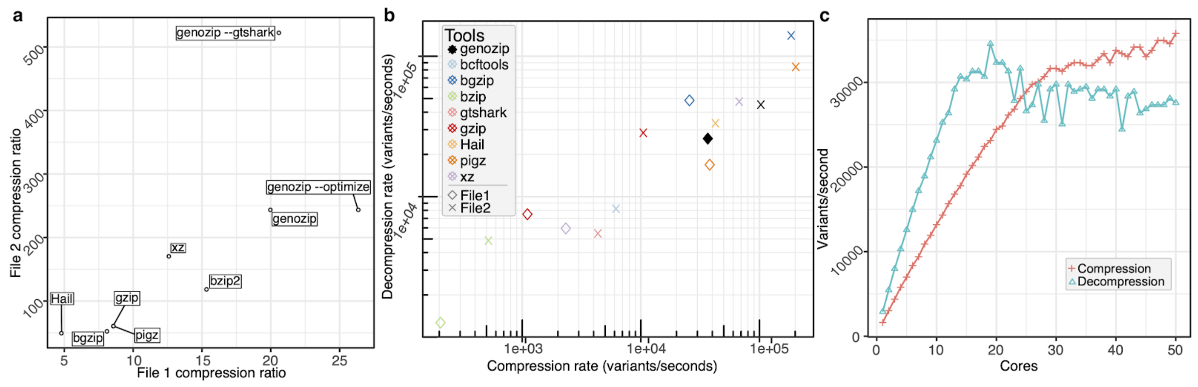


Fig. 1. Benchmarking `genozip` performance. (**a**) Compression ratios for `genozip` using three different options relative to five other commonly used compression tools (see labels) for two VCF files, the FORMAT-subfields-rich data (x-axis) and genotype-rich data dominant (y-axis). (**b**) Compression (x-axis) and decompression (y-axis) rates for `genozip` and five other tools on the two VCF files (see inset key), and the rates (**c**) `genozip` execution scalability with used CPU cores (see Supplementary Material).

Supplementary Information

for

genozip: a fast and efficient compression tool for VCF files

Divon Lan[1]*, Ray Tobler[1], Yassine Souilmi[1†], Bastien Llamas[1†]*

[1] School of Biological Sciences, The Environment Institute, Faculty of Sciences, The University of Adelaide, Adelaide SA 5005, Australia

[†] Equal contribution

* Corresponding authors: DL (divon.lan@adelaide.edu.au) and BL (bastien.llamas@adelaide.edu.au)

Content

SI.1. Full list of options of `genozip`, `genounzip`, `genocat` and `genols`

```
Compress VCF (Variant Call Format) files

Usage: genozip [options]... [files or urls]...

See also: genounzip genocat genols

Supported input file types: .vcf .vcf.gz .vcf.bgz .vcf.bz2 .vcf
                     .xz .bcf .bcf.gz .bcf.bgz
Note: for .bcf files, bcftools needs to be installed, and for
                     .xz files, xz needs to be installed

Examples: genozip file1.vcf file2.vcf -o concat.vcf.genozip
          genozip --optimize --password 12345 ftp://ftp.ncbi.nlm
                     .nih.gov/file2.vcf.gz

Actions - use at most one of these actions:
   -d --decompress   Same as running genounzip. For more details,
                     run: genounzip --help

   -l --list         Same as running genols. For more details,
                     run: genols --help

   -h --help         Show this help page. Use with -f to see
                     developer options.

   -L --license      Show the license terms and conditions for
                     this product

   -V --version      Display version number

Flags:
   -c --stdout       Send output to standard output instead of a
                     file

   -f --force        Force overwrite of the output file, or
                     force writing .vcf.genozip data to standard
                     output

   -^ --replace      Replace the source file with the result
                     file, rather than leaving it unchanged

   -o --output       <output-filename>. This option can also be
                     used to concatenate multiple input files
                     with the same individuals, into a single
```

```
                    concatenated output file

   -p --password    <password>. Password-protected - encrypted
                    with 256-bit AES

   -m --md5         Calculate the MD5 hash of the VCF file.
                    When the resulting file is decompressed,
                    this MD5 will be compared to the MD5 of the
                    decompressed VCF.
                    Note: for compressed files, e.g. myfile.vcf
                    .gz, the MD5 calculated is that of the
                    original, uncompressed file.

   -q --quiet       Do not show the progress indicator or
                    warnings

   -Q --noisy       Stop the suppression of warnings

   -t --test        After compressing normally, decompress in
                    memory (i.e. without writing the
                    decompressed file to disk) - comparing the
                    MD5 of the resulting decompressed file to
                    that of the original VCF. This option also
                    activates --md5.

   -@ --threads     <number>. Specify the maximum number of
                    threads. By default, this is set to the
                    number of cores available. The number of
                    threads actually used may be less, if
                    sufficient to balance CPU and I/O.
                    Tip: if you're concerned about sharing the
                    computer with other users, rather than
                    using --threads to reduce the number of
                    threads, a better option would be to use
                    the command nice, e.g. 'nice genozip....'.
                    This yields CPU to other users if needed,
                    but still uses all the cores that are
                    available

   --show-content   Show the information content of VCF files
                    and the compression ratios of each
                    component

Optimizing:
   -9 --optimize    Modify the VCF file in ways that are likely
                    insignificant for analytical purposes, but
                    make a significant difference for
                    compression. At the moment, these
```

```
                  optimizations include:
                  - PL data: Phred values of over 60 are
                  changed to 60.     Example: '0,18,270' ->
                  '0,18,60'
                  - GL data: Numbers are rounded to 2
                  significant digits.   Example: '-2.61618,-0
                  .447624,-0.193264' -> '-2.6,-0.45,-0.19'
                  - GP data: Numbers are rounded to 2
                  significant digits, as with GL.
                  - VQSLOD data: Number is rounded to 2
                  significant digits. Example: '-4.19494' ->
                  '-4.2'
                  Note: due to these data modifications,
                  files compressed with --optimized are NOT
                  identical as the original VCF after
                  decompression. For this reason, it is not
                  possible to use this option in combination
                  with --test or --md5

-B --vblock       <number between 1 and 2048>. Set the
                  maximum size of memory (in megabytes) of
                  VCF file data that can go into one variant
                  block. By default, this is set to 128 MB.
                  The variant block is the basic unit of data
                  on which genozip and genounzip operate.
                  This value affects a number of things: 1.
                  Memory consumption of both compression and
                  decompression are linear with the variant
                  block size. 2. Compression is sometimes
                  better with larger block sizes, in
                  particular if the number of samples is
                  small. 3. Smaller blocks will result in
                  faster 'genocat --regions' lookups

-S --sblock       <number>. Set the number of samples per
                  sample block. By default, it is set to 4096.
                  When compressing or decompressing a
                  variant block, the samples within the block
                  are divided to sample blocks which are
                  compressed separately. A higher value will
                  result in a better compression ratio, while
                  a lower value will result in faster
                  'genocat --samples' lookups

--gtshark         Use gtshark instead of the default bzlib as
                  the final compression step for allele data
                  (the GT subfield in the sample data).
                  Note: For this to work, gtshark needs to be
```

```
                    installed - it is a separate software
                    package that is not affiliated with genozip
                    in any way. It can be found here: https:/
                    /github.com/refresh-bio/GTShark.
                    Note: gtshark also needs to be installed
                    for decompressing files that were
                    compressed with this option.

One or more file names may be given, or if omitted, standard input
is used instead
```

```
Uncompress VCF (Variant Call Format) files previously compressed
with genozip

Usage: genounzip [options]... [files]...

See also: genozip genocat genols

Examples: genounzip file1.vcf.genozip file2.vcf.genozip
          genounzip file.vcf.genozip --output file.vcf.gz
          genounzip concat.vcf.genozip --split

Options:
   -c --stdout       Send output to standard output instead of a
                     file

   -z --bgzip        Compress the output VCF file(s) with bgzip.
                     Note: this option is implicit if --output
                     specifies a filename ending with .gz or .bgz.
                     Note: bgzip needs to be installed for this
                     option to work

   -f --force        Force overwrite of the output file

   -^ --replace      Replace the source file with the result
                     file, rather than leaving it unchanged

   -O --split        Split a concatenated file back to its
                     original components

   -o --output       <output-filename>. Output to this filename
                     instead of the default one

   -p --password     <password>. Provide password to access file
                     (s) that were compressed with --password

   -m --md5          Shows the MD5 hash of the decompressed VCF
                     file. If the file was originally compressed
                     with --md5, it also verifies that the MD5
                     of the original VCF file is identical to
                     the MD5 of the decompressed VCF.
                     Note: for compressed files, e.g. myfile.vcf.
                     gz, the MD5 calculated is that of the
                     original, uncompressed file.

   -q --quiet        Do not show the progress indicator or
                     warnings

   -Q --noisy        Stop the suppression of warnings
   -t --test          Decompress in memory (i.e. without writing
                     the decompressed file to disk) - comparing
                     the MD5 of the resulting decompressed file
                     to that of the original VCF. Works only if
                     the file was compressed with --md5

   -@ --threads       <number>. Specify the maximum number of
                     threads. By default, this is set to the
                     number of cores available. The number of
                     threads actually used may be less, if
                     sufficient to balance CPU and I/O.
                     Tip: if you are concerned about sharing the
                     computer with other users, rather than
                     using --threads to reduce the number of
                     threads, a better option would be to use
                     the command nice, e.g. 'nice genozip....'.
                     This yields CPU to other users if needed,
                     but still uses all the cores that are
                     available

   -h --help          Show this help page. Use with -f to see
                     developer options.

   -L --license       Show the license terms and conditions for
                     this product

   -V --version       Display version number

One or more file names must be given
```

Print VCF (Variant Call Format) file(s) previously compressed with genozip

Usage: **genocat** [options]... [files]...

See also: genozip genounzip genols

```
Options:
   -r --regions      [^]chr|chr:pos|pos|chr:from-to|chr:from-
                     |chr:-to|from-to|from-|-to[,...]
                     Show one or more regions of the file.
                     Examples:
                               genocat myfile.vcf.genozip -r22
                     :1000000-2000000  (A range of chromosome 22)
                               genocat myfile.vcf.genozip -r
                     -2000000,2500000-   (Two ranges of all
                     chromosomes)
                               genocat myfile.vcf.genozip -r21
                     ,22               (All of chromosome 21 and
                     22)
                               genocat myfile.vcf.genozip -r^MT
                     ,Y               (All of chromosomes except
                     for MT and Y)
                               genocat myfile.vcf.genozip -r^
                     -10000             (All sites on all
                     chromosomes, except positions up to 10000)
                     Note: genozip files are indexed
                     automatically during compression. There is
                     no separate indexing step or separate index
                     file.
                     Note: Indels are considered part of a
                     region if their start position is.
                     Note: Multiple -r arguments may be
                     specified - this is equivalent to chaining
                     their regions with a comma separator in a
                     single argument

   -t --targets      Identical to --regions, provided for
                     pipeline compatibility

   -s --samples      [^]sample[,...]
                     Show a subset of samples (individuals).
                     Examples:
                               genocat myfile.vcf.genozip -s
                     HG00255,HG00256    (show two samples)
                               genocat myfile.vcf.genozip -s
                     ^HG00255,HG00256   (show all samples except
                     these two)
```

```
                  Note: This does not change the INFO data
                  (including the AC and AN tags).
                  Note: sample names are case-sensitive.
                  Note: Multiple -s arguments may be
                  specified - this is equivalent to chaining
                  their samples with a comma separator in a
                  single argument

-G --drop-genotypes Output the data without the individual
                  genotypes and FORMAT column

-H --no-header    Do not output the VCF header

   --header-only  Output only the VCF header

   --GT-only      For samples, output only genotype (GT) data,
                  dropping the other subfields

   --strip        Do not output values for ID, QUAL, FILTER,
                  INFO; FORMAT is only GT (at most); Samples
                  include allele values (i.e. GT subfield)
                  only

-o --output       <output-filename>. Output to this filename
                  instead of stdout

-p --password     Provide password to access file(s) that
                  were compressed with --password

-@ --threads      Specify the maximum number of threads. By
                  default, this is set to the number of cores
                  available. The number of threads actually
                  used may be less, if sufficient to balance
                  CPU and I/O.
                  Tip: if you're concerned about sharing the
                  computer with other users, rather than
                  using --threads to reduce the number of
                  threads, a better option would be to use
                  the command nice, e.g. 'nice genozip....'.
                  This yields CPU to other users if needed,
                  but still uses all the cores that are
                  available

-q --quiet        Do not show warnings

-Q --noisy        Stop the suppression of warnings
-h --help         Show this help page. Use with -f to see
                  developer options. Use --header-only if
```

```
                    that is what you are looking for

   -L --license      Show the license terms and conditions for
                    this product

   -V --version      Display version number

One or more file names must be given
```

View metadata of VCF (Variant Call Format) files previously compressed with genozip

Usage: **genols** [options]... [files or directories]...

See also: genozip genounzip genocat

```
Options:
   -q --quiet        Do not show warnings

   -h --help         Show this help page

   -L --license      Show the license terms and conditions for
                     this product

   -V --version      Display version number
```

One or more file or directory names may be given, or if omitted, genols runs on the current directory

Options useful mostly for developers of genozip:

```
   --show-time       Show what functions are consuming the most
                     time

   --show-memory     Show what buffers are consuming the most
                     memory

   --show-sections   Show the section types of the output
                     genozip file and the compression ratios of
                     each component

   --show-alleles    Output allele values to stdout. Each row
                     corresponds to a row in the VCF file. Mixed-
                     ploidy regions are padded, and 2-digit
                     allele values are replaced by an ascii
                     character

   --show-dict       Show dictionary fragments written for each
                     variant block (works for genounzip too)

   --show-one-dict   <field-name>. Show the dictionary for this
                     field in a tab-separated list - <field-name>
                     may be one of the fields 1-9 (CHROM to
                     FORMAT) or a INFO tag or a FORMAT tag
                     (works for genounzip too)

   --show-gt-nodes   Show transposed GT matrix - each value is
                     an index into its dictionary

   --show-b250       Show fields 1-9 (CHROM to FORMAT) as well
                     as INFO tags - each value shows the line
                     (counting from 1) and the index into its
                     dictionary (note: REF and ALT are
                     compressed together as they are correlated.)
                     This also works with genounzip, but
                     without the line numbers.

   --show-one-b250   <field-name>. Show the values for this field -
                     may be one of the fields 1-9 (CHROM to
                     FORMAT) or an INFO tag

   --dump-one-b250   <field-name>. Dump the binary content of
                     this field, exactly as they appear in the
                     genozip format, to stdout - may be one of
                     the fields 1-9 (CHROM to FORMAT) or an INFO
                     tag
```

```
--show-headers    Show the sections headers (works for
                  genounzip too)

--show-index      Show the content of the random access index

--show-gheader    Show the content of the genozip header
                  (which also includes the list of all
                  sections in the file)

--show-threads    Show thread dispatcher activity

--debug-memory    Buffer allocations and destructions
```

SI.2. Implementation

`genozip` operates by segmenting the VCF file into separate sections defined by data type and appropriately processing each section, before applying a general purpose data compressor, `bzip2 (Seward, 1996)`, to each section. `genozip` executes a number of data transformations that take advantage of data covariance due to linkage disequilibrium, population structure, and potential lab biases, as well as non-textual relationships between numeric values in the file.

First, the VCF file is divided into *variant blocks* of up to 128 MB each (configurable with `--vblock`), and the samples within each variant block are further divided into *sample blocks* of up to 4,096 samples each (configurable with `--sblock`), from which the genotypes are extracted and transposed to create a *haplotype matrix*. Prior to compression, each haplotype matrix is further transformed by padding the ploidy to the maximal ploidy represented in the matrix, substituting 2-digit allele values with a single ascii character, and clustering the rows of haplotypes so that similar haplotypes are adjacent to one other. If the `--gtshark` option is used, clustering is skipped, and `GTShark` (Deorowicz and Danek, 2019) is used as the final-stage compressor of the *haplotype matrix*, instead of `bzip2`.

Second, the phase state data (i.e. | or /) are compressed – in the common case where the entire variant block has the same phase state, we drop the phase data entirely and just note the phase state in the variant block header.

Third, the data from each field (CHROM to FORMAT) and subfields of INFO and the sample data (as defined in the FORMAT field) are extracted into separate dictionaries, and their data are replaced with a dictionary index. An exception is the correlated REF and ALT fields that are combined into one field. For each field, a global dictionary is created for the entire file (or multiple files in case of concatenation), with new values added incrementally as each variant block is parsed, so that only a single pass is needed over the file, and crucially, the compressed file size grows sub-linearly with the number of VCF rows. For the first variant block, the dictionary entries are sorted by frequency, so that the highest frequency entries are efficiently encoded. The dictionaries for each field and the associated index data are then compressed separately. Index data from FORMAT subfields are compressed together as they are often correlated (for example, the DP and AD subfields). Dictionary search is implemented efficiently using hash tables, and an algorithm is run after the analysis of the first variant block to predict their size of the hash table for each field. This algorithm estimates the expected number of unique words of a particular field in the entire file from the gradient of the rate of appearance of new unique words within the first variant block. Extrapolating from the second derivative is obviously error prone, so an algorithm is in place for growing a hash table in run time, if its size was underestimated, while not affecting threads that are concurrently operating on it.

For the non-genotype indexed sample data we apply an additional optimization step of transposing the matrix prior to compressing it, to take advantage of experimental lab bias. In files with a large number of individuals, such as a File1 here, we have observed data differences between individuals that likely result from subtle differences in analysis tools used – for example, different floating point truncation conventions.

The POS field is often a large contributor to the overall entropy in single or small-sample files. To improve the compression of this field, we compress the difference between successive POS values rather than the POS value itself, thereby reducing the range of values and increasing compressibility.

SI.3. Compression ratio and speed benchmarks in more detail

To benchmark genozip's compression ratio compared to other popular and state-of-the-art compression tools, we used two different files from the 1000 Genome Project (1000 Genomes Project Consortium *et al.*, 2012; Sudmant *et al.*, 2015) that we refer to here as ‘File1’ and ‘File2’. We chose the two files for their substantial difference in their content characteristics (Table S1):

Table S1: Data content of File1 and File2

		File 1 VCF			**File 2 VCF**	
Allele values		6.1GB	7.1%		**30.2GB**	**49.2%**
Other sample data		**80.1GB**	**92.3%**		30.2GB	49.2%
Header and columns 1-9		0.5GB	0.6%		0.95GB	1.5%

File1: 1000 Genome Project phase 1 (The 1000 Genomes Project Consortium, 2012; chr1 ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/20110521/ALL.chr1.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz). The file contains 1,092 individuals, 3,007,196 variants, and “Other sample data” consisting mostly of the sample fields other than GT, and is the dominant data component in this file.

File2: 1000 Genome Project phase 3 ((Sudmant *et al.*, 2015); chr1 ftp://ftp.ncbi.nlm.nih.gov/1000genomes/ftp/release/20130502/ALL.chr1.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz). The file contains 2,504 individuals, 6,468,347 variants, and Allele values (i.e. the GT subfield) representing ~50% of this file. In this case, “Other sample data” is comprised of the phase state (/ or |) and the tab character that separates the samples – both of which are trivial in terms of compression. Therefore the allele values are about 97% of the remaining data content.

The differences in data content between these two files result in dramatically different compression ratios in all tools. In both cases, though, `genozip` achieves the best compression ratios (Table S2, Figure S1, Figure S2). `genozip` achieves the highest compression ratio amongst the all lossless compression tools, and offers competitive compression even compared to lossy tools such as `GTshark`.

`genozip` was tested in three ways - the first, is its default mode. The second, is with the `--optimize` option which modifies some data in the FORMAT and INFO subfields of VCF file in ways that are typically not significant for analytical purposes, but are quite significant for compression - namely, rounding some floating point numbers to two significant digits and capping some Phred values (see `genozip --help` for a detailed list). This compression by definition is not lossless. As can be seen in Table S2 `--optimize` significantly improves the compression of File1 that consists mainly for FORMAT subfield data, but no has no impact on File2 that has no FORMAT subfield data, The third, is using the `--gtshark` which utilizes `GTShark` for the final stage of compression of the genotype component of the VCF file, instead of the default `bzlib`. This significantly improves compression in File2 which is enriched in genotype data, but not as much in File1 that consists primarily of FORMAT subfield data.

We faced a number of challenges with the some of other compression tools:

`Hail` failed to decompress because it attempted to create very large intermediate files in the /tmp filesystem. This is a faulty software design as /tmp is typically quite small, and hence decompression of large files is bound to fail due to space constraints as happened in our case. To test a workaround and to allow at least partial inclusion of Hail (Hail Team) in this benchmark despite its malfunctioning, we chose File2 and used Hail's option to shard the decompressed file to many smaller files with its `parallel='separate_header'` option, and then concatenated the file together with the Linux `cat` command. The time shown is the combined time of `Hail` and `cat`.

`bcftools` failed to compress File1 - likely because it is a file created in 2011 prior to the latest versions of `bcftools`.

`GTShark` is not capable of processing FORMAT subfields, and hence is not capable of compressing File1.

`bcftools, Hail` and `GTShark` are not lossless - the decompressed file differs from the original.

Table S2: Compression ratio comparison

Tool	File 1 (MB)	vs. VCF		File 2 (MB)	vs. VCF
ORIGINAL	88,775	1X		62,728	
`genozip`	**4,430**	**21X**		**257**	**244X**
`genozip --optimize`	**3,360**	**26X**		**257**	**244X**
`genozip --gtshark`	**4,298**	**21X**		**120**	**523X**
`gzip`	10,282	9X		1026	61X
`bcftools -Ob`	Incapable	-		1007	62X
`bgzip`	10,873	8X		1187	52X
`bzip2`	5,767	15X		528	119X
`xz`	7,014	13X		367	171X
`pigz`	10287	9X		1030	61X
`gtshark`	Incapable	-		128	491X
`Hail`	18280	4.9X		1258	50X

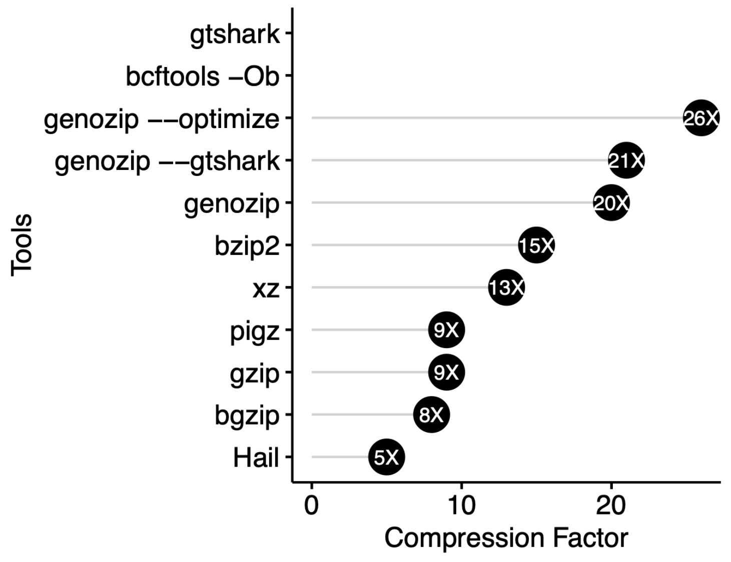


Figure S1: Compression factor for File1

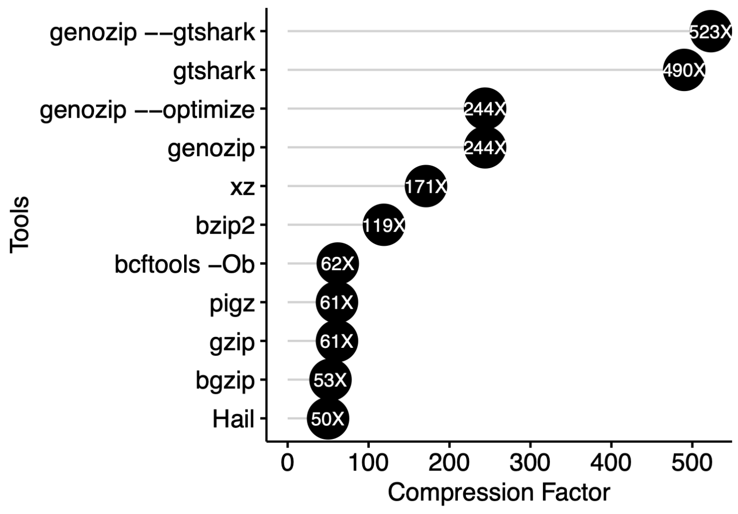


Figure S2: Compression factor for File2

In terms of execution time, genozip is designed to fully leverage the hardware available, unless explicitly restricted by the user. As such, it includes advanced memory and thread management components that allow almost linear scaling to tens of cores. In table S3, we have the execution time of each tool on our test machine that has 56 physical cores (4 X Intel® Xeon® Gold 6132 CPU @ 2.60GHz) and 755GB of usable memory, running an XFS file system with its default configuration on top of an SSD storage device. While generally multiple users have access to this computer, the benchmark was run one tool at time, and done so at a time when no other users or significant processes were running on the machine.

`bcftools`, `bgzip` and `xz` allow specification on number of threads, and were set to allow them to maximise the utilisation of the hardware - "`--threads 56`" for `bgzip` and `bcftools` and "`--threads 0`" for `xz`.

Table S3: Execution time comparison

Tool	**Compress**			**Decompress**	
	File 1	**File 2**		**File 1**	**File 2**
`genozip`	**1’22”**	**1’3”**		**1’56”**	**2’23”**
`gzip`	45’19”	10’17”		6’41”	3’47”
`bcftools`	N/A	17’27”		N/A	13’8”
`bgzip`	1’57”	35”		1’2”	46”
`bzip2`	244’30”	207’31”		39’33”	22’9”
`pigz`	1’19”	32”		2’58”	1’17”
`xz`	21’30”	1’36”		8’26”	2”16
`gtshark`	N/A	24’49”		N/A	19’42”
`Hail`	4’18”	2’32”[3]		N/A	3’14”

SI.4. Benchmarking of genotype-only compression algorithms

There are a number of algorithms published in recent years focused on compressing genotypes (allele values) within VCF files, while not being capable of compressing arbitrary VCF files. Some are also not capable of decompressing, and all do not guarantee lossless decompression.

Nevertheless, it is interesting to compare the performance of these algorithms on genotype data. In this benchmark we included comparing `genozip` in two modes – its default mode, and with the options `--gtshark -B2048` which would result in the best genotype-data-only compression. We compare against three genotype compression algorithms - `bgt` (Li, 2016), `GTC` (Danek and Deorowicz, 2018) and `GTShark` (Deorowicz and Danek, 2019). We also included the standard tools `gzip` and `bgzip` in this comparison, to appreciate how well all the genotype compression algorithms perform compared to generic compressors

To compare just the genotype data component of a VCF file, we started with File2 from our compression benchmark, and used the `--strip` option of `genocat` to strip out all data, except genotypes, CHROM, POS, REF and ALT fields, and set the FORMAT field to "GT":

```
genocat ALL.chr1.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.
3.vcf.genozip --strip > file2.stripped.vcf
```

In the results we can see that `genozip` in its default mode results in a better compression ratio of the stripped file than all tools except `GTShark` and `GTC`, while `genozip` with `--gtshark -B2048` is better than any other tool.

Table S4: Compression comparison of a genotype-only file

	Bytes	**Compression ratio**
Original: `file2.stripped.vcf`	64,956,779,894	
`genozip`	**201,369,011**	**323**
`genozip --gtshark -B2048`	**60,041,662**	**1082**
`gzip`	851,248,722	76
`bgzip`	939,705,276	69
`bgt`	298,990,428	217
`GTC`	138,182,645	470
`GTShark`	60,297,201	1077

SI.5. Core scalability test - raw data

To test the scalability of genozip with the number of available cores, we ran a compression and decompression test using File1 of our benchmark. We repeated the compression and decompression cycle scaling the number of used cores from 1 to 50 while recording the execution time (see Table S5). We observed that genozip compression scaled approximately linearly up about 28 cores, and then again about linearly up to 50 cores, but with a smaller slope. Decompression, on the other hand, scaled linearly up to about 20 cores after which adding additional cores had no benefit (see Figure 1b). A fundamental constraint on scaling is the need to access the disk file. In the case of genozip, the compressed file is between one and three orders of magnitude smaller than the original file, so it is the original file that is the constraint. We speculate that at least part of the difference in scaling between compression and decompression is the fact that SSD storage is faster in read operations (compression in our case) than write (decompression).

Table S5: execution time in core scalability test

Cores	Compress time (sec)	Uncompress time (sec)	genozip variants/sec	genounzip variants/sec
1	1,859	1,039	1,618	2,894
2	993	551	3,028	5,458
3	687	377	4,377	7,977
4	520	292	5,783	10,299
5	430	239	6,993	12,582
6	360	201	8,353	14,961
7	320	175	9,397	17,184
8	276	159	10,896	18,913
9	252	142	11,933	21,177
10	228	130	13,189	23,132
11	210	119	14,320	25,271
12	192	114	15,662	26,379
13	179	103	16,800	29,196
14	169	98	17,794	30,686
15	157	99	19,154	30,376
16	149	96	20,183	31,325
17	142	96	21,177	31,325
18	134	98	22,442	30,686
19	130	87	23,132	34,565
20	123	93	24,449	32,335
21	121	93	24,853	32,335
22	115	96	26,150	31,325
23	112	108	26,850	27,844
24	107	95	28,105	31,655
25	104	113	28,915	26,612
26	101	110	29,774	27,338
27	100	101	30,072	29,774
28	98	118	30,686	25,485
29	95	103	31,655	29,196
30	95	101	31,655	29,774
31	96	120	31,325	25,060
32	94	101	31,991	29,774
33	93	104	32,335	28,915
34	93	103	32,335	29,196
35	94	102	31,991	29,482
36	94	107	31,991	28,105
37	92	103	32,687	29,196
38	90	103	33,413	29,196
39	93	106	32,335	28,370
40	89	103	33,789	29,196
41	90	123	33,413	24,449
42	91	106	33,046	28,370
43	88	104	34,173	28,915
44	88	114	34,173	26,379
45	91	112	33,046	26,850
46	89	110	33,789	27,338
47	86	110	34,967	27,338
48	86	110	34,967	27,338
49	87	107	34,565	28,105
50	84	109	35,800	27,589

SI. References

Danek,A. and Deorowicz,S. (2018) GTC: how to maintain huge genotype collections in a compressed form. *Bioinformatics*, 34, 1834–1840.

Deorowicz,S. and Danek,A. (2019) GTShark: genotype compression in large projects. *Bioinformatics*, 35, 4791–4793.

Hail Team Hail.

Li,H. (2016) BGT: efficient and flexible genotype query across many samples. *Bioinformatics*, 32, 590–592.

Seward,J. (1996) bzip2 and libbzip2. *avaliable at http://www. bzip. org*.

Sudmant,P.H. *et al.* (2015) An integrated map of structural variation in 2,504 human genomes. *Nature*, 526, 75–81.

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

Chapter 2

Genozip - A Universal Extensible Genomic Data Compressor

Divon Lan[1,*], Ray Tobler[1,2], Yassine Souilmi[1,3,†,*], Bastien Llamas[1,2,3,†,*]

[1] Australian Centre for Ancient DNA, School of Biological Sciences, The Environment Institute, Faculty of Sciences, The University of Adelaide, Adelaide SA 5005, Australia
[2] Centre of Excellence for Australian Biodiversity and Heritage (CABAH), School of Biological Sciences, University of Adelaide, Adelaide, SA 5005, Australia
[3] National Centre for Indigenous Genomics, Australian National University, Canberra, ACT 0200, Australia

[†] Equal contribution
* Corresponding authors: DL (divon.lan@adelaide.edu.au), YS (yassine.souilmi@adelaide.edu.au), and BL (bastien.llamas@adelaide.edu.au)

Citation: Divon Lan, Ray Tobler, Yassine Souilmi, Bastien Llamas, Genozip: a universal extensible genomic data compressor, *Bioinformatics*, Volume 37, Issue 16, 15 August 2021, Pages 2225–2230, https://doi.org/10.1093/bioinformatics/btab102

Article history:
Received: 09 December 2020
Revision received: 25 January 2021
Editorial decision: 10 February 2021
Accepted: 12 February 2021

Published: 15 February 2021

Corrected and typeset: 17 May 2021

Statement of Authorship

Title of Paper	Genozip - A Universal Extensible Genomic Data Compressor
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	*Bioinformatics*, Volume 37, Issue 16, 15 August 2021, Pages 2225–2230, https://doi.org/10.1093/bioinformatics/btab102 **Received:** 09 December 2020 **Revision received:** 25 January 2021 **Editorial decision:** 10 February 2021 **Accepted:** 12 February 2021 **Published:** 15 February 2021 **Corrected and typeset:** 17 May 2021 5 citations as of 1/Feb/2022

Principal Author

Name of Principal Author (Candidate)	Divon Mordechai Lan		
Contribution to the Paper	Discovered the methods, developed the software, and wrote the initial version of the paper.		
Overall percentage (%)	90%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature	[illegible]	Date	20/04/2022

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Bastien Llamas		
Contribution to the Paper	Supervision, results interpretation and minor edits		
Signature		Date	20/04/2022

Name of Co-Author	Yassine Souilmi		
Contribution to the Paper	Supervision, results interpretation and minor edits		
Signature		Date	20/04/2022

Name of Co-Author	Raymond Tobler		
Contribution to the Paper	Supervision, results interpretation and minor edits		
Signature		Date	20/04/2022

Abstract

We present Genozip, a universal and fully featured compression software for genomic data. Genozip is designed to be a general-purpose software and a development framework for genomic compression by providing five core capabilities – universality (support for all common genomic file formats), high compression ratios, speed, feature-richness, and extensibility.

Genozip delivers high-performance compression for widely-used genomic data formats in genomics research, namely FASTQ, SAM/BAM/CRAM, VCF, GVF, FASTA, PHYLIP, and 23andMe formats. Our test results show that Genozip is fast and achieves greatly improved compression ratios, even when the files are already compressed.

Further, Genozip is architected with a separation of the Genozip *Framework* from file-format-specific *Segmenters* and data-type-specific *Codecs*. With this, we intend for Genozip to be a general-purpose compression platform where researchers can implement compression for additional file formats, as well as new codecs for data types or fields within files, in the future. We anticipate that this will ultimately increase the visibility and adoption of these algorithms by the user community, thereby accelerating further innovation in this space.

Availability and implementation
Genozip is written in C. The code is open-source and available on GitHub (https://github.com/divonlan/genozip). The package is free for non-commercial use. It is distributed as a Docker container on DockerHub and through the conda package manager. Genozip is tested on Linux, Mac, and Windows.

Supplementary information
Supplementary data are available at Bioinformatics online.

1 Introduction

Genomic data production is growing rapidly as sequencing prices continue to drop, making data storage and transfer a core issue for researchers, healthcare providers, service facilities, and private companies. To date, most users have relied upon compression software that implements the RFC 1951 format (Deutsch, 1996); e.g., gzip (Gailly and Adler, 2010), bgzip (Li, 2011b) and others), a general-purpose compression format that was designed decades ago and is not specifically tailored for genomic data.

Many novel algorithms have emerged in recent years that effectively compress one or more of the data types embedded in genomic files (e.g., GTShark (Deorowicz and Danek, 2019) and SPRING (Chandak *et al.*, 2019)). However, these algorithms are typically implemented within a rudimentary software package that inadvertently lacks the breadth of features required for a software to be useful in many real-world use cases; most importantly, most work with only one of the common file formats. These limitations have meant that none of these software packages are currently widely used by the genomic researcher and practitioner community.

Here we introduce a new version of the compression software Genozip, which has been nearly completely re-written from an earlier version designed to compress VCF files (Lan *et al.*, 2020). Genozip now offers five core capabilities:

1. Universality - Genozip supports all common genomic file formats - FASTQ, SAM/BAM/CRAM, VCF, GVF, FASTA, PHYLIP, and 23andMe.
2. High compression ratios - better than all other universal tools tested.
3. Speed - in most cases, faster than other tools.
4. Feature richness - providing an array of features that allow integration into pipelines, specification of compression options, and development tools to allow developers to extend Genozip easily.
5. Extensibility - with a clear separation of the Genozip *Framework* from the file formats being compressed and from the codecs used for compression, it is fairly easy to add support for more file formats as well as new codecs to improve compression of

specific data types of any specific fields within genomic files.

2 Software description

Genozip provides a command line interface that consists of four commands: `genozip` for compression, `genounzip` for decompression, `genocat` to display or subset a compressed file, and `genols` to show metadata associated with the compressed files.

Genozip is currently optimised to compress FASTQ, FASTA, SAM/BAM/CRAM, VCF/BCF, GVF, PHYLIP, and 23andMe files, including files that are already compressed into *.gz*, *.bz2,* or *.xz* formats. However, Genozip can also compress any other file format. Compression of .cram, .bcf, or .xz files requires the software packages samtools, bcftools, or xz, respectively, to be available in the PATH environment variable. Genozip allows multiple files of identical or different formats to be specified in the command line. Files that share a common format can be *bound* together with `genozip --output` and subsequently unbound with `genounzip --unbind`. This functionality is beneficial for packaging a large number of samples together for delivery or archiving.

Genozip can be integrated into analytical pipelines in two ways. First, `genozip` and `genounzip` may be used with pipes. Second, `genocat` provides random-access to user-specified sections of a `.genozip` file and facilitates file subsetting. When using `genocat` to subset files, the targeted data are identified using the `--samples option` for VCF files and the `--regions` option for SAM, VCF, FASTA, GVF, and 23andMe file types. `--downsample` downsamples any file type. Further, because `.genozip` files are indexed during data compression, a separate indexing step is not required.

In addition, `genocat` offers built-in file format *translation*, and currently offers *translations* between SAM and BAM, from SAM or BAM to FASTQ, between FASTA and PHYLIP and from 23andMe to VCF, using `genounzip`'s `--bam, --sam, --fastq, --phylip, --fasta` and `--vcf` options, respectively.

Genozip offers a range of data integrity and security options. To support data security requirements that comply with ethical standards now expected for modern genomic projects, Genozip allows encryption of the data using the `--password` option. With this option, data are encrypted with the standard Advanced Encryption Standard (AES) algorithm (Fips, 2009), using the strongest mode available (256 bits). To ensure data integrity, Genozip includes a built-in MD5 (Rivest, 1992) option triggered by using `--md5` or `--test`. This calculates (in `genozip`) or verifies (in `genounzip` and `genocat`) the MD5 sum of the source data on the fly and stores it within the compressed genozip file. This MD5 sum is then viewable using `genols`.

Genozip offers two lossless compression modes: `--best,` which is the default and results in the highest compression ratio, and `--fast,` which optimises compression speed at the cost of somewhat reduced compression ratios (see Supplementary Information section 12). While Genozip is strictly lossless by default, a lossy `--optimise` (or `--optimize`) option is also offered, which further improves compression by modifying the data in ways that typically do not impact downstream analysis (See Supplementary Information section 3).

Additionally, Genozip supports compression with or without a reference genome sequence. Providing a reference improves compression of the sequence data component in SAM/BAM/CRAM, FASTQ, and VCF files. A reference file may be generated from a FASTA file with `genozip --make-reference` and used with `genozip --reference` or `--REFERENCE`. The latter option stores information from the reference within the resulting compressed file, obviating the need to provide the reference as a separate file during the decompression step. Including the reference information within the compressed file is particularly useful when binding several genomic data files together for delivery.

Finally, fine level information on various aspects of the data compression can be accessed by the user via the large suite of `--show` options (see Supplementary Information section 8). For instance, `--show-stats` provides compression statistics broken down by data type within the file. We anticipate that such information will be insightful for end-users and particularly useful when developing new compression algorithms.

3. Methods

3.1 Framework and architecture

The Genozip framework (Figure 1) interprets the user's command line, reads the source genomic file (referred to as the *txt file*), and divides it into *vblocks*. Each vblock comprises a certain number of full *txt file* lines, limited by size that is determined by the user with the `--vblock` option (default: 16MB). By default, a *line* means an actual ASCII line in the *txt file*; however, this is flexible—e.g., for FASTQ, a *line* comprises four textual lines, and for BAM it comprises one alignment record.

Once the Genozip framework has read the *vblock txt data* into memory using its main thread (called the *I/O thread*; Figure 1), a separate *compute thread* is spawned to segment the *vblock*. This segmentation step is followed by the final compression step that ultimately generates *z data*, which is the final compressed data for the *vblock*. When the compression step is completed, the compute thread terminates, and the compressed *vblock* is handed back to the *I/O thread* that appends it to the `.genozip` compressed file being generated on disk.

3.2. The segmentation step

A *segmenter* is a module that is specific to the file format being compressed. Genozip currently has nine segmenters, one each for FASTQ, FASTA, SAM, BAM, VCF, GVF, PHYLIP, 23andMe, and Generic. If samtools (Li *et al.*, 2009) is also installed, the SAM segmenter can also handle CRAM files by reading them as SAM. The *Generic* segmenter handles all other file formats for which genozip does not have a *segmenter* in a default manner. Importantly, interested parties can add more segmenters to Genozip in the future.

The *segmenter* is called by the Genozip framework to work on one line of *txt data* at a time, and the job of the *segmenter* is to segment this line into its individual data components, store these in *contexts* (which are described in detail in Supplementary Information section 2), and declare how each context should be handled in the compression stage.

The segmenter starts by breaking up the *txt line* into the top-level data fields and deciding what to do with each data field. Broadly, it has six options:

1. Placing the data directly in its appropriate *context*. We refer to a short string of data inserted into a context as a *snip*. Each new *snip* encountered by the Genozip framework is added to a *dictionary* within each *context*, and an index is added to the *dictionary* entry in a data buffer for this *context* called the *b250 buffer.* Accordingly,

the `.genozip` file stores each *snip* only once and uses a numeric index to point to it throughout the file.

2. Further segmenting a field into its subfields: Rather than making a *snip* of the entire field data as it appears in the file, the segmenter can insert a special *snip* type called a *Container*, which defines the structure of the data of this field, where the data itself is stored in other *contexts* that are named in the container. *Containers* can define records containing multiple types of data, as well as arrays of similar data elements or arrays of records. The entire *vblock* is described as a single *Container snip* placed in the TOPLEVEL *context*.

   This is a key feature that enables the decompressor to be generic. Indeed, in most cases, the decompressor need not have any built-in awareness of the details of each file format. The file format structure is encoded in the data itself, and a *vblock* may be reconstructed by traversing the data starting from the TOPLEVEL.

3. Exploiting known relationships between fields, subsequent lines, and/or external data to improve the compression. For that, the segmenter may define *contexts* as needed — for example, it may store multiple fields in a single context or may decompose a field into multiple contexts. It can be as simple as exploiting a mathematical relationship between fields, but it can also be complex - for example, the sequence data in FASTQ and SAM are aligned to a reference if the `--reference` option is used.

4. Using one of the Genozip's framework built-in algorithms. Some relationships occur frequently, for which Genozip has built-in algorithms. These include the *seg_pos* algorithm that exploits the nearness of position data in subsequent lines, if it exists and *seg_id* algorithm that handles ID data that starts with an alphabetical prefix followed by a number (such as "rs23424") as well as LOOKUP and DELTA vs. another field on the same line or vs. the same field in a previous line or vs. the pair file (in case of paired-end FASTQ files). Details about these built-in algorithms can be found in Supplementary Information section 2.

5. Preparing the data for a *specific* codec. Rather than inserting a *snip*, the segmenter can store the data of a field in the *local* buffer of the *context* in any proprietary way, in preparation for consumption by a *specific* codec in the compression stage.

6. Declaring a *context* to be an *alias.* There are cases where multiple fields contain data with similar characteristics, in which case storing them in a single *context* can improve compression. To achieve this, we can define a *context* as an *alias* of another, essentially sharing their data. For example, in SAM format, there are multiple Optional tags that express data in CIGAR format (MC:Z, OC:Z, and others), which are all defined as aliases of a *context* named `@CIGAR`.

In the *Generic* segmenter used for unrecognized file formats, the segmenter is trivial and does not actually segment the data - instead, the entire *vblock* data is placed in a the *local* buffer of a single context.

A detailed example of how these six options work, as well as a full list of how each of the nine segmenters in Genozip handles each data field appears in Supplementary Information section 2.

3.3. Context management

Segmentation step: Each *vblock* maintains its own set of *contexts* – the set consisting of one *context* for each data component. A context is a data structure that includes the *dictionary*, *b250,* and *local* data buffers as well as additional information.

Context merging step: We maintain one global set of similar *contexts* within an object called the *z_file* to which we merge vblock *contexts'* dictionary data after the segmentation is completed for a *vblock*, thereby incrementally creating a global dictionary containing, in a particular *z_data context*, all values of that appear for that data component in the entire file (except for singletons - see Supplementary Information section 2).

Cloning step: When a new *vblock* is created, the current dictionary and related information of each *context* are *cloned* from the *z_file* to the new *vblock* by the framework.

Writing step: After the compute thread terminates and the *vblock* is handed back to the I/O thread, the I/O thread writes the *vblock*'s *z_data* (containing *b250 and local* sections) to the output `.genozip` file. The merged dictionary data is written upon the completion of computing of all *vblocks*.

Context cloning, concurrent *dictionary* access and *context* merging in a multi-threaded environment are difficult, and doing so with minimal synchronisation between threads to avoid a bottleneck that would limit scaling CPU cores, is even more so. We employ advanced multi-threading mechanisms that ensure that all threads can operate on the same dictionaries concurrently while minimising the use of synchronisation objects like mutexes, minimising memory copies, and ensuring O(1) dictionary lookups, uniqueness of dictionary entries, and thread-safety. Details of how this is done are in Supplementary Information section 6.

3.4 The compression step

Within the *compute thread* of any specific *vblock*, and once the segmentation is complete for all lines and the contexts dictionaries have been merged back into *z_file*, the framework proceeds to compress the two buffers of each *context* present in this vblock—namely, the *b250* and the *local* buffers. Each buffer is compressed with one of the available codecs. There are two types of codecs in Genozip:

Generic codecs - these are lzma (Pavlov, 2007), bz2 (Seward, 1996), bsc (http://libbsc.com/), and *none*. The first three are standard codecs for which Genozip utilises a modified version of the standard libraries, and the fourth is a codec that essentially keeps the data as-is.

Specific codecs - these are additional codecs that compress a specific data type better than the generic codecs and would often be *complex* codecs—which means that they will perform some processing of the data, and then complete the compression by applying one or more of the built-in codecs. *Specific* codecs can be added to compress any specific field of any genomic file format.

For the *b250* and *local* buffers of each *context*, the codec is selected automatically by sampling approximately 100KB of the buffer data in the first *vblock* in which this *context* is first encountered, and compressing it with each of the four codecs. The best codec is selected by an algorithm that chooses the codec with the best compression ratio unless the compression ratio between the best two codecs is close enough, and the execution time is different enough, in which case it selects the faster codec of the two. Subsequent *vblocks* use the same codec and need not test again. In `--fast` mode, a modified selection algorithm is used that is biased towards speed even at the expense of a small difference in compression.

A segmenter may specify a codec for the *local* buffer of any particular *context*, overriding the automatic selection. In the segmenters provided, we use this privilege only when we set the codec to a *specific* codec.

Genozip currently has four *specific* codecs:

A. *acgt* - used for compression of a sequence of nucleotides, which is expected to contain mostly, but not necessarily exclusively, 'A', 'C', 'G' or 'T' characters. It is used to compress FASTA sequence data and characters (bases) from the SEQ field of FASTQ and SAM file formats that are not mapped to a reference.

B. *domqual* - used for compression of a string of Phred quality-scores in SAM and FASTQ formats, in the common case where there is one dominant score value.

C. *hapmat* - used for compression of a matrix of haplotypes derived from FORMAT/GT fields in VCF. The algorithm is described in (Lan *et al.*, 2020) and has been re-implemented to serve as a codec.

D. *gtshark* - triggered by the `--gtshark` option, utilises the software package GTShark (Deorowicz and Danek, 2019) as a codec for the same haplotype matrix as *hapmat* as an alternative to *hapmat*. This was already implemented in (Lan *et al.*, 2020), where we have shown it to be significantly better but significantly slower than *hapmat* for the FORMAT/GT data component in VCF files that have a large number of samples. It has been re-implemented as a codec for FORMAT/GT on top of the new framework and with a new fast in-memory (rather than disk-based) communication channel between `genozip` and `gtshark`. This is an example of how Genozip can be easily extended to incorporate new codecs for specific data types.

More details on the algorithms for each of these codecs can be found in the Supplementary Information section 6.

3.5 Compressing against a reference

Genozip does not require a reference but takes advantage if one is available to better compress FASTQ, SAM/BAM, and VCF data.

To use a reference with Genozip, a *Genozip reference file* must first be created using `genozip --make-reference`. This is a one-time step for any particular reference FASTA file. The Genozip reference file creation is implemented by segmenting the reference FASTA data with a specialised segmenter, which generates a Genozip file containing a pre-processed version of the reference data in a format that is readily usable by Genozip, as well as hash tables for use of the Genozip Aligner, indexing data and additional metadata.

When using a particular Genozip reference file to compress data for the first time, Genozip generates two cache files. These files are used to accelerate the loading of the reference data and the Genozip Aligner hash tables in subsequent executions of Genozip, and may be deleted if such acceleration is not needed. The acceleration is achieved by loading the cache files, if they exist, using the operating system's paging system rather than libc allocated memory, allowing portions of the reference data to be paged-in as needed, and also enables sharing of the loaded pages between concurrently running Genozip processes, resulting in reduced memory consumption and instantaneous loading in the case of concurrent Genozip instances.

The VCF segmenter uses reference data to avoid storing REF and/or ALT data and referring to the reference if possible. Since the REF and ALT fields usually represent only a small fraction of the information content of a VCF file, the gains are modest, however.

The SAM and BAM segmenters use reference data in two different ways, depending on whether the *txt line* being segmented is aligned (i.e., contains values in the RNAME, POS, and CIGAR fields) or not, and the FASTQ segmenter uses the reference similar unaligned SAM/BAM:

1. For an aligned SAM/BAM/CRAM *txt line*, the segmenter decomposes the data into three *contexts*: `SQBITMAP`, `NONREF`, and `NONREF_X`. `SQBITMAP` is a bitmap consisting of a bit for every base in the sequence that "consumes a reference", as defined in the SAM specification (https://samtools.github.io/hts-specs/SAMv1.pdf page 8) according to the CIGAR string. The bit is set to 1 if the base is the same as the base in the reference data at its position. If not, the bit is set to 0, and the base character is placed in `NONREF`. Bases in the sequence that "do not consume a reference", according to the CIGAR string, are also placed in `NONREF`. `NONREF` is set to be compressed with the *acgt* codec that requires a second *context* for the

CODEC_XCGT data, which is `NONREF_X` (see Supplementary Information section 6).

2. For an unaligned SAM/BAM/CRAM *txt line* and a FASTQ sequence line, the Genozip Aligner is used. It utilises the same three *contexts* described above and two additional ones: `GPOS` and `STRAND`. The Aligner algorithm (see Supplementary Information section 4) finds the position in the reference to which the sequence string at hand best aligns. This algorithm is extremely fast as it does not attempt to find the biologically correct alignment, just one that compresses well. The aligner determines the location in the reference, using a coordinate called *gpos* (Global Position) - which is a single 32-bit unsigned integer covering the entire reference genome, and indicates whether it is forward or reverse complement relative to the sequence (which we call *strand*). The segmenter then stores the *gpos* and *strand* in the *local* buffers of the `GPOS` and `STRAND` contexts, respectively (the strand is stored as a bitmap with '1' meaning forward) and proceeds to populate the `SQBITMAP` and `NONREF` contexts as before, based on whether or not each base in the sequence matches the corresponding base in the forward or reverse complement reference.

3.6 Indexing

While Genozip is designed as a compression tool rather than a data analysis tool, it also contains some capabilities that allow direct integration into analysis pipelines. Chief among these, is indexing of the data done by the Genozip framework during segmentation, which then allows subsetting the data with the `genocat --regions` option: a Segmenter may notify the Genozip framework of the chromosome (or contig) and position of each line being segmented. As the segmentation progresses, the framework collects data per vblock - namely, it records which chromosomes appear in the vblock, and the minimum and maximum position of each chromosome within the vblock. These data are then emitted to the generated compressed genozip file as the SEC_RANDOM_ACCESS section.

During `genocat --regions`, vblocks that contain no data from the requested regions are skipped entirely, while vblocks that do contain data from the requested regions are decompressed, but only lines that are included in the requested regions are emitted.

In addition, Genozip reference files are also indexed in the same way, so when subsetting a file that requires a reference (i.e. the `--reference` option is used), Genozip only reads the vblocks of the reference file that overlap with the regions requested.

Currently, the segmenters for VCF, SAM, BAM, GVF and 23andMe implement this capability.

This indexing method is more coarse-grained than the BGZF-block level indexing that is common in standard indexes of genomic file formats, as subsetting requires decompression of entire vblocks (16MB of txt data in the default configuration) vs just BGZF blocks (64KB of data), and hence subsetting is significantly slower. However, in practice, this may be sufficient for many analysis applications.

4 Results

We evaluated the performance of Genozip by compressing genomic files as they most commonly appear in real-world research and clinical situations - namely, already compressed in fastq.gz, BAM, CRAM, and vcf.gz formats. Regarding CRAM, we tested two different commonly used versions of CRAM files - a version containing the same data as the BAM file and a version optimised by binning quality data. For VCF, we tested a single-sample file. We previously reported the compression performance of multi-sample VCF using an earlier version of the HapMat codec in (Lan *et al.*, 2020). For BAM, CRAM, and FASTQ, we also tested with Genozip's `--optimise` option.

The FASTQ, BAM, and VCF files (Table 1 and Supplementary Information Table S10) were obtained from a public FTP server of the National Center for Biotechnology Information (NCBI), while the CRAM files were generated from the BAM file using Scramble (Bonfield, 2014) with the highest compression ratio (-9 option) and, in addition, for the binned-quality CRAM, with the quality-binning option -B (Table 1). The reference file used was based on a modified version of GRCh37 as required by the particular BAM file tested (see Supplementary Information section 12) and was prepared with: `genozip --make-reference $grch37-fasta-file`.

Genozip improved the compression of these already-compressed files in every scenario we tested by a 1.2–5.7 factor (Figure 2 as well as Table S11 in the Supplementary Information).

In addition, we performed tests comparing Genozip's compression ratio on raw (uncompressed) files (Table S8 in the Supplementary Information), as well as compression and decompression time, to several popular tools. These additional results can be found in the Supplementary Information section 12 and illustrated in Figure 3, Table 2 and Supplementary Information Table S9. Again, in all cases tested, Genozip outperformed other software for compression ratio by a 1.3-4.4 factor, while also faster than other tools in most, but not all, cases.

Table 1: Files used for testing against already-compressed files. These files are of five formats commonly used in research and clinical settings. We demonstrate that Genozip can significantly improve the compression for each of these files. See details of these files in Supplementary Information Table S10

File type	**File size**	**Genozip command** `--optimise` added for the Optimised test
.fastq.gz	3.6 GB (R1+R2)	`genozip --pair $file-R1 $file-R2 -e $ref-file`
.bam	147 GB	`genozip $file -e $ref-file`
.cram (lossless)	102 GB	`genozip $file -e $ref-file`
.cram (binned)	79.5 GB	`genozip $file -e $ref-file`
.vcf.gz	128 MB	`genozip $file -e $ref-file`

Table 2: Raw-file benchmark results. Results of compression of uncompressed genomic files with genozip and other commonly used tools for each file format. More details are available in Supplementary Information Table S9

Tool	Ratio	Compress time	Decompress t.
VCF			
pigz	15.9	1.9 sec	3.1 sec
bcftools	11.7	23.82 sec	21.02 sec
bzip2	25.3	260.05 sec	43.37 sec
genozip	33.6	7.1 sec	6.53 sec
SAM			
pigz	3.4	00:12:40.3	00:34:17.4
samtools	3.2	00:23:16.7	00:29:48.5
scramble -9	4.7	00:27:58.4	00:17:34.4
genozip -e	5.8	00:33:41.1	00:27:55.3
Optimized cram: scramble -9B	6.0	00:48:56.1	00:19:10.4
Optimized genozip -9	7.6	00:30:51.1	00:20:38.0
FASTQ			
pigz	4.2	00:14:34.5	00:34:17.4
bwa mem | samtools sort | scramble -9	5.4	03:42:54.0	00:48:24.7
genozip -e	6.8	00:16:40.1	00:08:31.7
genozip -9e	18.6	00:08:52.3	00:05:26.4

5 Conclusion

Genozip provides not only excellent compression for raw (uncompressed) genomic files, but also provides excellent compression when applied directly to already-compressed genomic files, as is common in real-world applications. Genozip is also universal and works on all common genomic files, uniquely so amongst currently available genomic file compressors.

Further, by providing a modular and extensible architecture, Genozip is also a framework that can be used for rapid development and deployment of new compression algorithms for established or emerging genomic data types and file formats.

Funding

D.L. is supported by a scholarship from the University of Adelaide. Y.S. is supported by the Australian Research Council (ARC DP190103705). R.T. is an ARC DECRA fellow (DE190101069). B.L. is an ARC Future Fellow (FT170100448).

Fig.1. Genozip high-level architecture. The Genozip framework interprets and reads the input file(s) in the main thread (I/O thread) and divides them into vblocks, which are then segmented. Segmentation is followed by the compression step. Compressed vblocks are sent back to the I/O thread to create the .genozip output(s).

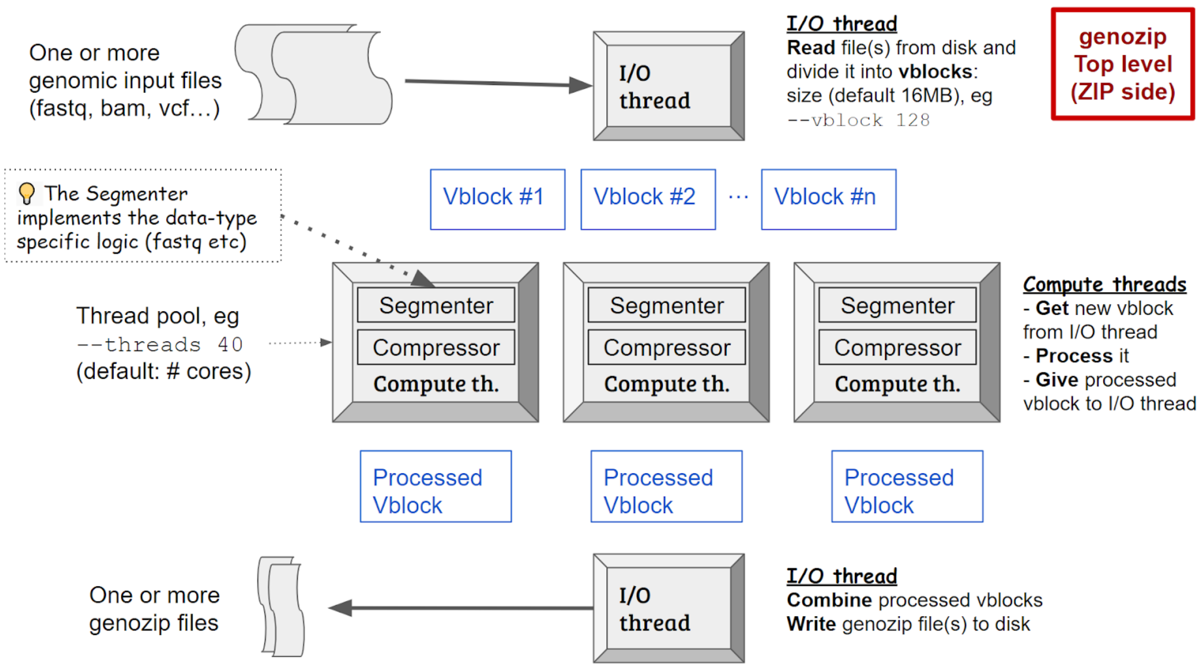

One or more genomic input files (fastq, bam, vcf…)
I/O thread
I/O thread
Read file(s) from disk and divide it into vblocks: size (default 16MB), eg
--vblock 128
genozip Top level (ZIP side)
The Segmenter implements the data-type specific logic (fastq etc)
Vblock #1
Vblock #2
…
Vblock #n
Thread pool, eg
--threads 40
(default: # cores)
Segmenter
Compressor
Compute th.
Compute threads
- Get new vblock from I/O thread
- Process it
- Give processed vblock to I/O thread
Processed Vblock
One or more genozip files
I/O thread
I/O thread
Combine processed vblocks
Write genozip file(s) to disk

Fig. 2. Sizes of Genozip-compressed files relative to already-compressed source files. The blue bars represent the source files (see Table 1), with the corresponding file extensions at the bottom. The orange and grey bars are for Genozip compression with the default, lossless mode and the `--optimise` option, respectively. See also results in Supplementary Information Table S11.

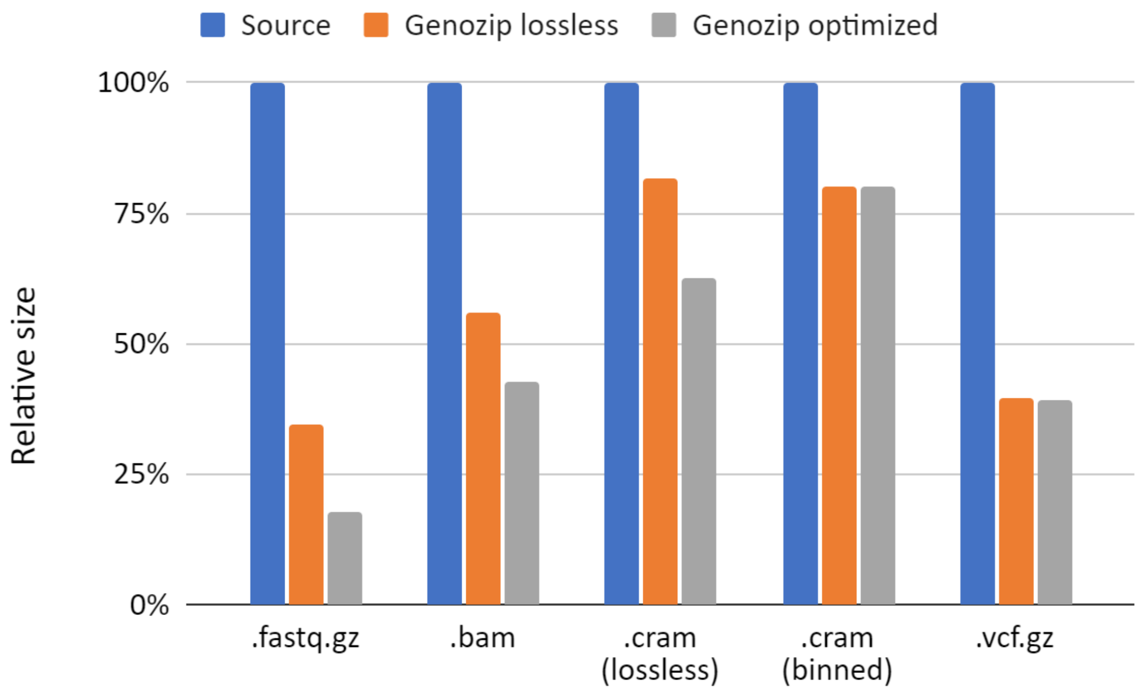
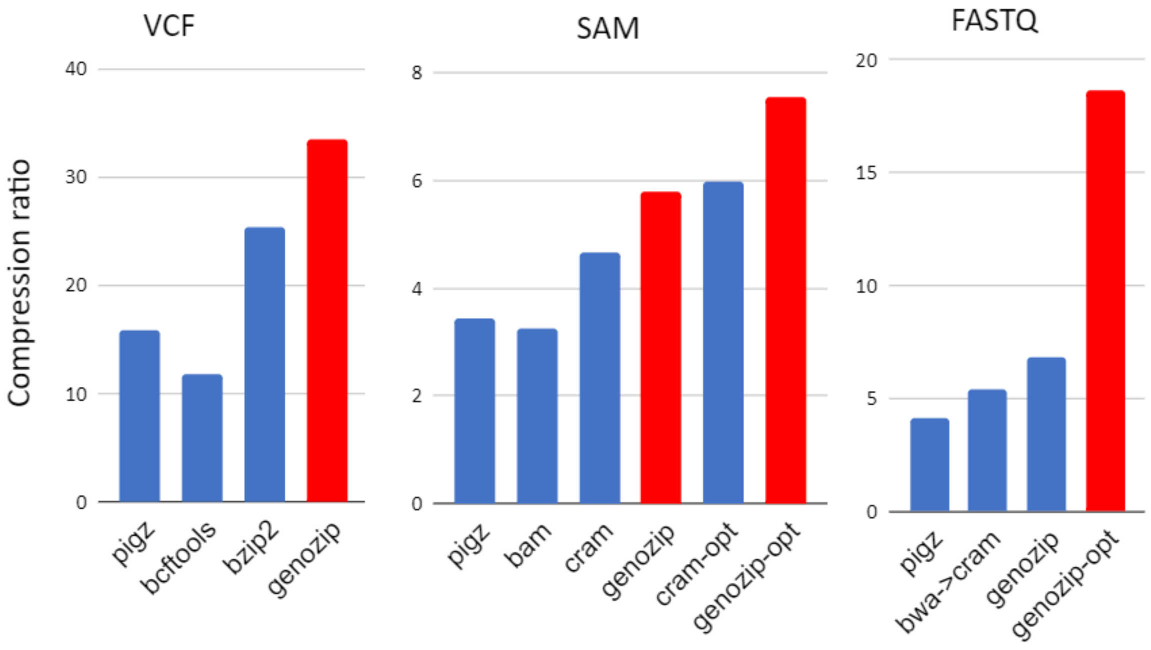


Fig. 3. Raw (uncompressed) files benchmark results. The three panels show compression ratios of various relevant compression formats indicated at the bottom relative to uncompressed VCF (left), SAM (middle) and FASTQ (right) files relative. See Supplementary Information section 12 for more details.



References

Bonfield,J.K. (2014) The Scramble conversion tool. *Bioinformatics*, **30**, 2818–2819.

Chandak,S. *et al.* (2019) SPRING: a next-generation compressor for FASTQ data. *Bioinformatics*, **35**, 2674–2676.

Deorowicz,S. and Danek,A. (2019) GTShark: genotype compression in large projects. *Bioinformatics*, **35**, 4791–4793.

Deutsch,P. (1996) DEFLATE Compressed Data Format Specification version 1.3 RFC Editor.

Fips,P. (2009) 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, US Department of Commerce, November 2001. *Link in: http://csrc. nist. gov/publications/fips/fips197/fips-197. pdf*.

Gailly,J.-L. and Adler,M. (2010) GNU gzip.

Lan,D. *et al.* (2020) genozip: a fast and efficient compression tool for VCF files. *Bioinformatics*, **36**, 4091–4092.

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

Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Pavlov,I. (2007) Lzma sdk (software development kit).

Rivest,R. (1992) RFC1321: The MD5 Message-Digest Algorithm RFC Editor, USA.

Seward,J. (1996) bzip2 and libbzip2. *available at http://www.bzip.org*.

Genozip - A Universal Extensible Genomic Data Compressor

Supplementary Information

Divon Lan[1,*], Ray Tobler[1,2], Yassine Souilmi[1,3,†,*], Bastien Llamas[1,2,3,†,*]

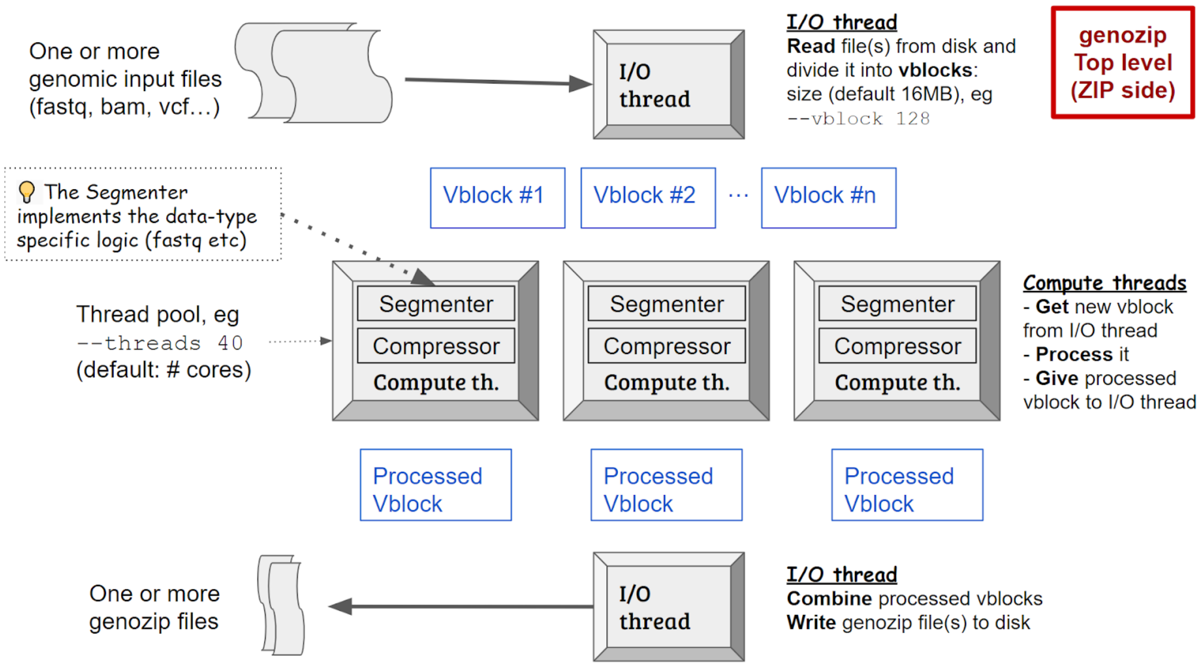
[1] Australian Centre for Ancient DNA, School of Biological Sciences, The Environment Institute, Faculty of Sciences, The University of Adelaide, Adelaide SA 5005, Australia
[2] Centre of Excellence for Australian Biodiversity and Heritage (CABAH), School of Biological Sciences, University of Adelaide, Adelaide, SA 5005, Australia
[3] National Centre for Indigenous Genomics, Australian National University, Canberra, ACT 0200, Australia

[†] Equal contribution
* Corresponding authors: DL (divon.lan@adelaide.edu.au) and BL (bastien.llamas@adelaide.edu.au)

TABLE OF CONTENTS

SI.1. Genozip high level architecture

Figure S1 (same as Figure 1 in main text) - Genozip high-level architecture. The Genozip framework interprets and reads the input file(s) in the main thread (I/O thread) and divides them into vblocks, which are then segmented. Segmentation is followed by the compression step. Compressed vblocks are sent back to the I/O thread to create the .genozip output(s).


One or more genomic input files (fastq, bam, vcf…)
I/O thread
I/O thread
Read file(s) from disk and divide it into vblocks: size (default 16MB), eg --vblock 128
genozip Top level (ZIP side)
The Segmenter implements the data-type specific logic (fastq etc)
Vblock #1
Vblock #2
…
Vblock #n
Thread pool, eg --threads 40 (default: # cores)
Segmenter
Compressor
Compute th.
Compute threads
- Get new vblock from I/O thread
- Process it
- Give processed vblock to I/O thread
Processed Vblock
One or more genozip files
I/O thread
I/O thread
Combine processed vblocks
Write genozip file(s) to disk

At a high level, Genozip is a C language program tested to run on Linux, MacOS and Windows (64bit). The same C program may be invoked by 4 command line commands: *genozip*, *genounzip*, *genocat*, and *genols*. The first two have major code components associated with them, described below. *genocat* essentially executes *genounzip* where the output goes to stdout and flags useful in pipeline analysis are made available. *genols* is lists the genozip files in a disk directory along with some metadata.

Here we will describe the ZIP side (i.e. the compression side). The PIZ side (decompression) follows a very similar architecture, where the inverse operations are carried out as expected.

The system was designed so that the logic related to compression of specific data elements is mostly coded in the ZIP side, while the PIZ side is as generic as possible. The reason for this is to allow the compressor to evolve, adding compression algorithms to an ever growing list of data elements, without needing to change the decompressor or the file format. There are a few exceptions to this that will be discussed.

When run, a Genozip process consists of one main thread (called the *I/O thread*) that is responsible for parsing the user's command line, reading and writing files to disk, as well as splitting the data stream into *vblocks* and spawning additional threads (called *compute threads*) to do the CPU-intensive compression or decompression work.

On the ZIP side, a *compute thread* consists of two main stages:

- First to run is the *Segmenter*: this parses the uncompressed file data (called *txt data*) first into individual logical lines (that are actual ASCII lines in all supported formats, with the exception of FASTQ where every 4 ASCII lines are counted as one logical line), and then parsing each line into its components, and finally applying a compression algorithm specific to the type of each component. The output of these component compression algorithms is stored in data structures called *contexts*. The Segmenter is where most of the logic related to specific data types (SAM, FASTQ etc) is located, while the rest of the Genozip code is not data-type specific.

- Second is the *Compressor*: it compresses each context using the appropriate codec that can be a *generic* or *specific* codec, as listed in the main text.

On the PIZ side, a *compute thread* does the inverse:

- First to run is the *Decompressor,* which decompresses the *sections* as read from disk to recreate the *contexts*

- Second to run is the *Reconstructor* that is data-type specific, querying the various contexts to losslessly reconstruct the original data. The instructions of which algorithms to use for reconstruction are part of the data themselves and not hard coded in the *Reconstructor* (with some exceptions) — i.e., the *Reconstructor* effectively pulls data from contexts and applies various algorithms as directed by instructions in the data. Upon completion, the *txt data* is reconstructed in the *vblock*, ready to hand back to the *I/O thread*.

Each *compute thread* receives a single *vblock* from the I/O thread, processes it, and hands it back to the *I/O thread* upon completion. *Compute threads* run in parallel and might complete out-of-order. The *I/O thread* receives back the processed *vblocks*, writes them to disk in the correct order, with each *context* within the *vblock* written as one or more *sections* in the final Genozip output file. Genozip utilises as many processor cores as it can, unless the user limits the number of threads with `--threads`. A substantial part of Genozip consists of thread synchronisation algorithms that are designed to ensure maximum CPU core scalability with minimum bottlenecks.

SI.2. The Segmenter

General

This section expands and adds details to the segmenter description in the main paper.

A *segmenter* is a module that is specific to the file format being compressed. Genozip currently has eight segmenters, one each for SAM, VCF, FASTQ, FASTA, GVF, PHYLIP, 23andMe, and Generic. More segmenters can be added to the open source code by interested parties.

The *segmenter* is called by the framework with one line of *txt data* at the time, and the job of the *segmenter* is to segment this line into its individual data components, store these in *contexts* which will be described hereinafter, and declare how each context should be handled during the compression stage.

The segmentation process from the Segmenter's viewpoint

We now explain how a segmenter works and how the framework interacts with one by walking through a simple example of a single *txt line*, and explain in detail the logic related to it. As an example, we use the following VCF line (the VCF header line is provided here only for clarity) and the VCF segmenter.

```
CHROM POS ID    REF ALT QUAL FILTER INFO              FORMAT Smp1    Smp2
chr12 0   rs123 G   A   100  PASS   AC=1;AN=2;AF=0.5 GT:DP  1|1:37 1|0:32
```

The first step for the VCF segmenter is to break the *txt line* into the top-level data fields, which are separated by a tab character in the case of VCF. This is a trivial task using macros provided by the framework.

Next, the VCF segmenter needs to decide what to do with each data field. Broadly, it has six options:

1. The segmenter may <u>place the data directly in its appropriate *context*</u>. This is the simplest case, and indeed the most common one. In the case of the *txt line* above, the VCF segmenter applies this strategy to the CHROM, QUAL, FILTER

fields—placing them in the CHROM, QUAL and FILTER *contexts* respectively.

The Genozip framework, in turn, would add these *snips*, to a *dictionary* within each *context*, if they are not already in this *context's* dictionary, and would place an index to the *dictionary* entry in a data buffer for this *context* called the *b250 buffer.* This way the Genozip file stores each *snip* only once, and uses a numeric index to point to it throughout the file.

2. The segmenter may <u>further segment a field into its subfields</u>, and then for each subfield recursively apply one of these six options. In this particular *txt line*, the VCF *segmenter* segments the INFO field to its subfields, with each INFO tag being considered a field. The structure of the INFO field, including the number of subfields the tag prefixes (eg "AC=") is represented in a data structure called a *Container*. This *Container* is created for this INFO field, and placed as a *snip* in the INFO *context*. The *Container* contains only the description of each *item* which together the *record* that is the INFO field (not the values themselves), so that *txt lines* that have the same tag names in the same order in their INFO field will have the same INFO *snip,* and hence the same index placed in the *b250* buffer. The VCF segmenter will now need to consider the values of the AC, AN and AF fields ("1", "2" and "0.5") and recursively apply one of these six options to each of them.

   A *Container* can also contain an array—in our example, the entire set of samples in a VCF line goes into one *Container* that describes the tags (GT and DP in this case), as well as the number of *repeats* (= the number of samples; in this case, 2)—and is placed in the SAMPLES *context*. As before, the GT and DP values are not included in the *Container*, and the VCF segmenter needs to recursively choose one of these six options for each of them.

   The GT field is further segmented into its individual haplotype values as well as its phasing value ('|' in this case). The VCF segmenter encodes this as a *Container* which contains an array where *repeats*=ploidy, and each array entry has a single item—the haplotype. The phase value is treated as the separator between repeats, which is also defined in the *Container* data. This *Container*, once again not containing the haplotype values themselves, is placed in the GT *context*.

   Similarly, Optional fields in SAM are an array of records, and any specific Optional field that is an array (i.e. SAM type 'B') is itself stored as a *Container* that is the array.

The INFO field in VCF and the ATTRS field in GVF are a *Container* containing a record, with the items being the tags, etc.

All segmenters are required to have a single *Container* per *vblock* that goes into the TOPLEVEL *context*. This *Container* describes the entire *vblock*—it is an array with *repeats*=number of lines in the *vblock*, and the items describe the structure of a line as defined for this file format. In the case of our example *txt line:* the TOPLEVEL has 10 items (CHROM, POS, ID, REF+ALT, QUAL, FILTER, INFO, FORMAT, SAMPLES and EOL), where INFO and SAMPLES are themselves *Container*s as described above.

This is a key feature in enabling the decompressor to be generic. Indeed, the decompressor need not have any built-in awareness of the details of each file format because the file format structure is encoded in the data itself, and a *vblock* may be reconstructed by traversing the data starting from the TOPLEVEL.

3. The segmenter may <u>exploit known correlation between fields</u> in order to improve the compression. In the case of the *txt line* above, the VCF segmenter employs this strategy in two occurrences:

    a. Since the REF and ALT field are highly correlated, they are stored together ("G A" in this *txt line* example) in a single context REF+ALT.

    b. In the case of AC, AN, AF, the segmenter checks whether AC equals AN*AF as expected with the common use of these tags in VCF files. If this is indeed the case, the AN and AF fields are stored normally in the AN and AF *contexts* while the AC is simply stored as the *snip* "SPECIAL AC". Since normally we would expect all AC values in a *vblock* to be AN*AF, the AC dictionary will contain in this case only a single entry "SPECIAL AC" and the *b250* data for each *vblock* will have an entry for each line in the *vblock* for which we have these INFO tags, but all values will be the same: the *dictionary* index of the "SPECIAL AC" snip. This will cause the *b250* data to compress to a trivial size in the compression step.

       Using SPECIAL AC will then require providing an extension to the uncompress side called a *special reconstructor* for AC. `genounzip`, when encountering the "SPECIAL AC" snip in the *b250* of the AC *context*, will call the AC *special reconstructor* that will implement the specific special reconstruction algorithm, in this case simply emitting the value of AN*AF.

Typically, we have a handful of special reconstructors for each file format that represent opportunities to compress based on relationships between fields or even use of external information.

Important to note, it is not an error if AN*AF is not equal to AC. It is not the role of Genozip to enforce correctness of field values and it is always tolerant if the value is not as expected. In this case, it will simply store the value of AC in the AC context instead of “SPECIAL AC”.

SPECIAL algorithms may be as simple as multiplying AN and AF, or may be as complex as needed. For example, we use the same mechanism to analyse the SEQ field (nucleotide sequence data) in SAM and FASTQ files against an external reference file.

SPECIAL algorithms are a powerful tool for encoding any type of relationship between fields, and as such, may contribute significantly to the compression ratio. However, this power comes at a cost, namely that the reverse algorithm for retrieving the original value from the encoded value in combination with the value of the related fields, must be encoded as a *SPECIAL reconstructor* on the decompressor side, thereby adding data-type-specific code to the decompressor side, which we are attempting to minimize.

4. The segmenter may: <u>use one of the genozip’s framework built-in algorithms</u>. In our *txt line* example, we have two occurrences of this strategy:

    a. The POS field uses the built-in *seg_pos* interface. This inspects the POS value compared to the previous line.If the absolute value of the difference is at most 32000, it stores the snip “DELTA (*this_pos - prev_pos)*” in the POS *context*. If it is more than 32000, it stores “LOOKUP” snip in the POS context’s *dictionary* / *b250*, and the value of POS itself as a 32 bit unsigned integer in the POS context’s *local* buffer. `genounzip`, when encountering a “LOOKUP” snip, reconstructs the value from the *local* buffer.

   b. The ID field uses the built *seg_id* interface. This attempts to split a *snip* into an alphabetical and a numeric part—“rs” and “123” in our example. The *snip* stored in the *context*’s *dictionary* would be “LOOKUP rs” and the unsigned integer value 123 will be stored in the *context*’s local buffer.

5. The segmenter may <u>prepare the data for a *specific* codec</u>. The segmenter may store the data of a field in any proprietary way, in preparation for consumption by a *specific* codec in the compression stage. In our example, the VCF segmenter stores the haplotype data in a haplotype matrix stored in this *vblock* data structure, which will be later compressed using the *hapmat* or *gtshark* codecs.

6. The segmenter may <u>declare the field to be an *alias.*</u> Sometimes it is beneficial to store more than one field in a single *context*. For example, the INFO/END tag (not in our *txt line* example) is normally an integer with a value between this line’s and the next line’s POS value. Therefore, we alias INFO/END with the POS context, which combined with the *seg_pos* interface normally creates a series of DELTA *snips* with delta values smaller than either POS or INFO/END would have on their own.

A full list of how each of the eight segmenters in Genozip handles each data field appears in the Table S4**.**

<u>**Pre-segmentation *Cloning* and post-segmentation *Merging***</u>

When the *segmenter* for a particular *vblock* is complete, the framework merges each *context* created in this *vblock*, with the corresponding global *context* held in memory in an object called the *z_file*. This merge does not affect the *b250* and *local* data that remains private to the *vblock*, and is focused mostly on the *dictionary* data. Since segmentation of many *vblocks* happens in parallel in multiple threads, each thread adding *snips* to their private *dictionary* fragment, with the possibility of multiple parallel threads adding some identical snips to their respective dictionary fragments, careful merger of the dictionary fragments into the *z_file* global dictionary, as well as re-writing the indices in the *vblocks*’ *b250* buffer is required.

When the segmentation of a new *vblock* begins, the global dictionary is *cloned* from the *z_file* to the local *context* in the new *vblock*. For efficiency, no memory is actually copied but rather a set of parameters is set to determine which dictionary entries within the *z_file* dictionary are available to each *vblock*. This synchronisation algorithm is written carefully so that merging of completed *vblocks* into the global dictionaries may occur at the same time as active segmenters in other threads are accessing the very same dictionaries, and the access by segmenting threads can be done without the use of synchronisation objects like mutexes, which would create a bottleneck and limit the scaling to a large number of cores.

Singleton detection

The framework also includes a singleton detector: if a *snip* appears for the first time in a particular *vblock*, and appears only once in this *vblock*, then it is placed in the context's *local* buffer instead of in the *dictionary* and *b250*. This way, we avoid bloating global dictionaries with singletons and keep singletons local to the context. This is important, since *vblock* memory is freed once the output data of this *vblock* is written to disk, while the global dictionaries remain in memory until compression of the entire file is complete.

Hash tables & the Snip Diversity Estimation Algorithm

When a *segmenter* calls the framework to enter a *snip* into a context, the framework first needs to lookup that *snip* a the dictionary to know whether it is new and should be added to the context's local dictionary fragment, a new index generated, and that new index added to the local *b250* data, or whether this *snip* already exists, in which case the existing dictionary index should be added to the local *b250*. To perform this lookup in O(1), each context also contains a hash table that allows rapid lookup. To achieve O(1), it is necessary for the target range of the hash function, and hence the initial size of the hash table, to be proportionate with the total number of distinct *snips* across the entire file for any particular *context*. This varies drastically between *contexts*, and indeed, it can be very large for contexts for which we expect millions of distinct snips (resulting in a hash table size of tens of MBs of RAM or more) , or very small (in case the file has only a handful of unique values of a particular field) for contexts like FORMAT in VCF.

The challenge we face, is to estimate, without reading the entire file, how many unique values exist in the entire file for each particular context. For this we have developed the *snip diversity estimation algorithm*:

In most *contexts*, many new snips appear early in the file, but as we progress in the file, since many of the snips were already observed before, less new snips are encountered. To estimate the expected total number of snips of a particular *context*, we analyse the data for this *context* for the first *vblock* in which this *context* is encountered. We look at the first derivative within this *vblock*:the density of new *snips* within a context—d(new_snips)/d(lines), as well as the second derivative—d(density)/d(lines). We then estimate the total file size, which might not be known if the file is compressed (with gzip, bzip2, xz, bgzip, bam or bcf) or if it is piped from stdin. We use the values of the first and second derivative and the estimated file size to estimate the number of unique *snips* across the entire file. Performing a simple mathematical integral to reach the results yields a poor match to the real values, and hence we enhance this with several heuristics based on observations of real world data. The full algorithm can be found in the function `hash_get_estimated_entries()`.

The Snip format

Snips stored by a Segmenter in a context may be simply the text of the data field itself; indeed, this is most often the case. However, Genozip has a number of snip opcodes that allow storing the data in a more compressible format, where applicable. In this case, rather than storing the data string as the snip, we construct a snip starting with one of the opcodes in the table below (each being one byte), followed by the required parameters. The decompression side has built-in algorithms for reconstructing the data field's string based on the data in the snip, while the *b250* array consisting of these snips is expected to be less random and/or with a smaller dictionary, and hence compress better than if we were to insert the data itself.

Table S1: Snip opcodes. When a snip has one of these values as its first byte, it is reconstructed as prescribed in this table, rather than just copying the snip.

Name	**Parameters**	**Reconstruction algorithm**
LOOKUP	*prefix* (optional)	*prefix* followed by the next value from *local*
OTHER_LOOKUP	*other_context* *length* (optional)	The value from *local* of *other_context*. If local is of type LT_SEQUENCE, then use *length* characters.
PAIR_LOOKUP	-	Copy value in the matching row in the paired file (when compressing FASTQ with `--pair`)
CONTAINER	*structure*	Recursively reconstruct the values from the contexts listed in structure, and combine them as specified
SELF_DELTA	*delta*	Value on previous line + *delta* (*delta* may be negative)
OTHER_DELTA	*other_context* *delta*	Last value from *other_context* + *delta*

PAIR_DELTA	*delta*	Value of matching row in paired file + *delta*
SPECIAL	*algorithm* *params (optional)*	Reconstructor to use the requested *algorithm* with *params*

The *Context* data structure

Genozip achieves its flexibility relative to file formats, by compressing individual data components of files into *contexts*, which are based on a recursive data format called *snips* which allows arbitrarily complex component-specific logic. It is recursive in the sense that *snips* might themselves be containers containing other snips.

Each context contains three data *buffers*:

1. The *dictionary.* This buffer is generated as the txt file is segmented, containing a single entry for each snip that appeared so far in the file. When a *vblock* segmentation commences, the dictionaries of all contexts, as updated by previously completed vblocks, are cloned into this vblock as are accessed on a read-only basis. If new snips are discovered in this vblock that are not already in their respective dictionary, vblock-private dictionary fragments are created. When a vblock segmentation completes, these fragments are integrated back into the global dictionary. Care is taken to make sure the global dictionary contains exactly one entry per snip, even though multiple vblocks running in parallel might discover the same snip and add it to their respective dictionary fragments.

2. The *b250.* This buffer contains 32-bit indices into the dictionary of all the snips of this *context* in a particular *vblock*, in the order they will be read by the decompressor. If no *b250* buffer exists, the decompressor will take the data from *local* (see below), but if it does exist, it must contain exactly one entry for each related data component in the txt file. To improve compression of the b250 buffer, some 8-bit values (instead of 32 bit) are used in some cases: A. if the snip the most, 2nd most or 3rd most frequent snip (as measured in the first vblock in which this context is used) B. If the snip index is one higher than the index of the previous snip in this b250 (this will result in a highly compressible run in some cases) C. missing non-GT values in VCF samples.

3. The *local* buffer. This buffer contains data that is private to this *vblock* and is not in the dictionary. Some contexts use the *local* buffer to contain singleton snips that are

expected to appear, as determined by a heuristic algorithm, only in a single *vblock*, while some contexts use the *local* buffer to store all the data, when this data is expected to be mostly private to this *vblock*, rather than using a dictionary. A context may utilise *local* to store either *snips* or alternatively simple data, such as integers or bitmaps.

This context based data structure is extremely flexible because it is independent of any particular genomic file format: when coding a segmenter for a particular file format we may pick and choose the most appropriate algorithms for each context, or develop new ones if needed, and *genozip* as a system may evolve fast in the future, by easily updating algorithms for specific contexts. Indeed, *genozip* can also serve as a good testing platform for new algorithms that focus on specific data elements of genomic data, by allowing creating or modifying contexts these specific elements.

We illustrate this by describing four contexts: POS (as it appears in VCF, SAM and 23andMe), ID (as it appears VCF, 23andMe, GVF), XA tag in SAM and Compound Field. A full list of all algorithms follows.

As explained above, a snip is created by the Segmenter (compressor side) is usually reconstructed as-is by the Reconstructor (decompress side), unless it begins with one of the special opcodes, which may be followed by parameters.

The built-in *POS* algorithm

The POS algorithm is one of the built-in framework algorithms that segmenters may use. It is designed for numeric fields that contain a 32-bit unsigned value, and have the property that subsequent lines tend to have values that are quite near each other. This is a characteristic of the fields that are a coordinate within a specific chromosome, and hence the name. For example, this is the case for the POS field in VCF, the POS and PNEXT fields in SAM, the POS field in 23andMe. It also appears in various optional fields.

This is a good example of how a particular algorithm, in this case one designed to handle POS data, works with the three context buffers.

In this example, let’s assume we have 3 lines in a particular vblock of a txt file (for example a VCF or SAM file), with POS values of 1000, 1500 and 10000000:

Table S2: SELF_DELTA example. Example the contents of the *b250* and *local* buffers of a POS *context*, after segmenting the values 1000, 1500, and 10000000

Value	***dictionary***	***b250***	***local***
1000	"LOOKUP"	0	1000
1500	"SELF_DELTA 500"	1	
1000000		0	1000000

As we can see, 1000 is the first POS value in this *vblock*. Since this is the first POS value, it will be stored as an unsigned 32bit integer in *local*, and the snip "LOOKUP" is added to the dictionary—telling the decompressor to lookup the value in local. Finally, the value 0, the index of the snip "LOOKUP" in the dictionary, is added to *b250*. 1500, the second POS value, is encoded as a delta vs. the previous line. Hence we add the snip "SELF_DELTA 500" to the dictionary, store the dictionary index of this snip, 1, in *b250*, and nothing in *local.* The third POS value, 1000000, is deemed to be too distant from the previous value 1500—beyond the defined threshold which is 32000—to be worthy of a delta. It is therefore stored as a "LOOKUP" snip. We already have a "LOOKUP" in the dictionary, so we needn't add another one: just place its index, 0, in our *b250* and the value, 1000000, in *local*.

The built-in *ID* algorithm

This is another build-in algorithm the framework provides segmenters.

Genomic data often contains IDs that are structured as a string containing a letter prefix, followed by a numeric suffix. Examples include the ID field in VCF, Dbxref attribute in GVF and EnstID identifiers that often appear in GVF attributes.

Table S3: ID example. Example the contents of the *b250* and *local* buffers of an ID *context*, after segmenting the values "rs999", "strange_id" and "rs123"

Value	***dictionary***	***b250***	***local***
rs999	"LOOKUP rs"	0	999
strange_id	"strange_id"	1	
rs123		0	123

In this example, rs999 is the first ID value in this *vblock*. It is separated to its numeric component, 999, which is stored in local, and its letter component, rs, which is combined with LOOKUP to create the snip “LOOKUP rs”. This instructs the decompressor to output “rs” followed by the value looked up from *local*. Finally, the dictionary index of this snip, 0, is placed in *b250*.

The second ID value, strange_id, does not comply with our assumption regarding the format of IDs, namely being composed of letters followed by numeric characters. We therefore stored it as a simple snip “strange_id”, with the dictionary index of this snip, 1, placed in *b250*. This demonstrates the general approach of the various context algorithms: a specific algorithm is designed to optimise the compression based on assumed data format, but the algorithm can always handle data which is not compliant to the format, as long as the general file format rules (as defined in the file format specification—e.g., the VCF or SAM) are not violated.

The third ID value, rs123, is similarly decomposed with the index of the already existing “LOOKUP rs” snip, 0, placed in *b250* and the numeric value in *local*.

<u>The built-in *Container* algorithm</u>

A Segmenter may define *Container* snips.

We saw some examples of *Container* snips above. Here we take a closer look at how a *Container* snip is formed.

A *Container* snip is one that contains 0 or more *repeats* of a collection of items, collectively called a record:

- All elements of an array have the same structure—each is a record of items
- The items are defined by their *context*
- Each item within a record might be of a different type and its values go into a specific context.
- Each item may have prefix—the same prefix is used for this item in all records
- Each item and the entire record might have a one or two character separator. The same separator is used for all records.

Note that the *Container* snip only defines the structure of data; the values themselves of each item are stored in their respective contexts. These values may themselves be *Container* snips, enabling the ability of genozip to define data formats recursively.

Let's look at an example—the SA:Z optional tag in SAM:
`SA:Z:chr1,1000,+,151M,10,2;chr2,2000,-,151M,10,2`

This tag is defined in REF as as an array of records:
"SA:Z:(rname ,pos ,strand ,CIGAR ,mapQ ,NM ;)+"

In this case of SA, the *Container* snip will look like this:
(prefix="SA:Z:", repeats=2, (@RNAME,','), (@POS,','), (@STRAND,','), (CIGAR,','), (@MAPQ,','), (NM:i,';'))

This *Container snip* contains a prefix, and 2 repeats of 6 items each. The first 5 items have a ',' separator and the 6th has a ';' separator. The decompressor reconstructing this snip will reconstruct the data by querying these six contexts (@RNAME, @POS, @STRAND, CIGAR, @MAPQ and NM:i), twice for each, as well as insert the prefix separators in the appropriate places.

In this case, @RNAME, @POS, @STRAND and @MAPQ are contexts that are shared between the SA:Z, OA:Z and XA:Z tags in SAM. CIGAR is shared with the primary CIGAR field of SAM, an NM:i is shared with the NM:i tag in SAM.

The *Container snip* logic is extremely flexible in its ability to represent different types of data. The number of items in a *Container snip* as well as the number of repeats and the separators is in no way fixed; indeed, every individual *snip* can be defined as needed.

The built-in *Compound Field* algorithm

A segmenter may use this algorithm to decompose a string value into logical components, by breaking it at predefined separators. The number of subfields is variable, and each occurrence may have a different number of subfields.

This context algorithm is used for the QNAME field in SAM and the Description lines in FASTQ and FASTA.

The *Compound Field* is built on top of a *Segmented* snip: it creates *contexts* for each subfield, and results in a *Segmented* snip with one record and all subfields. The Container snip itself is stored in the main context, while each subfield is stored in its own subfield context.

Let's look at an example: a QNAME field in the first two lines of a SAM file:

```
A00488:21621:1078
A00488:21766:1078
```

When processing each one of these values, the *Compound Field* algorithm splits this string by the separator which is ':' (colon) in this case. Each component is then placed in its ordinal Q*NAME context, with * being 0 for the first component, 1 for the second etc (we used the numerals 0-9 followed by A-Z). It then places a *Container snip* in the QNAME context:

The first value `A00488:21621:1078` causes 4 contexts to update:

Q0NAME.dictionary ⟵ 'A00488'   Q0NAME.b250 ⟵0 (index into the dictionary)
Q1NAME.dictionary ⟵ '21621'   Q1NAME.b250 ⟵0
Q2NAME.dictionary ⟵ '1078'   Q2NAME.b250 ⟵0

QNAME.dictionary ← CONTAINER*(repeats=1, (Q0NAME,':'), (Q1NAME,':'), (Q2NAME, ''))'*
QNAME.b250 ← 0 (index into the dictionary)

The second value `A00488:21766:1078` causes 4 contexts to update:
Q0NAME.b250 ← 0 (identical to previous line, index into an existing snip in the dictionary)
Q1NAME.dictionary ← 'SELF_DELTA 145'   Q1NAME.b250 ←1 (delta vs previous line)
Q2NAME.dictionary ← 'SELF_DELTA 0'   Q2NAME.b250 ←1 (delta vs previous line)
QNAME.b250 ← 0 (*Container snip* is identical to previous line, i.e., same index)

Note that in the common case where the entire file has QNAME data of the same format, we will have only one one item (one *Container snip*) in the QNAME context dictionary, and the entire QNAME b250 will be a run of 0's, compressing to a trivial size. Similarly, the other components also typically compress very well either because they are frequently identical between consecutive lines, or a small delta between consecutive lines.

The built-in *Special Snip* mechanism

In most contexts, the Segmenter (i.e. compression side) forms snips with one of the built-in algorithms, therefore, no code is required on the Reconstructor (i.e. decompress) side that is specific to this context. This gives us flexibility of evolving genozip by improving how we compress various contexts by coding the Segmenter only, allowing older genozip decompressors to still correctly reconstruct the new files.

However, in a few cases the desired algorithm is specific to the data component at hand, and cannot be generalised—as described above for the INFO/AC field in VCF. In these cases, we create a "*Special snip*" that redirects the Reconstructor to execute a special algorithm. This requires code in both the Segmenter and Reconstructor.

A *Special snip* contains the ID of the algorithm, and optionally parameters of the algorithm. Two C functions implementing the logic of any particular special snip must be provided: one for segmenting during, and the other for reconstructing during decompression, and the special snips must be declared in the header file of their data type (eg vcf.h, sam.h etc).

When a Special Snip algorithm is defined by a Segmenter, the Segmenter need not use it for all lines. Indeed, for any particular data component, the segmenter may choose to store a Special Snip for some lines and it may store data directly for other lines. This is commonly done when we *expect*, based on our knowledge that a value of a field may be expressible by a formula of data in other fields and/or external data. The segmenter will check for any particular line, whether our formula is indeed correct for the data of that line—and if it is, store the Special Snip providing the information needed by the Special Snip's reconstructor function to reconstruct the data—or whether our formula is not correct for this line, in which case the segmenter can just store the data as is.

Full list of contexts used, by file type

Table S4: List contexts by data (file) type. Each data type as a default set of contexts listed in this table. Additional contexts may be created to store optional fields (INFO and FORMAT fields in VCF, Optional fields in SAM/BAM etc).

Name	Snip	*local* use
SAM		
RNAME	As-is. Used for both RNAME and RNEXT fields.	-
QNAME	Container: Compound field	-
Q?NAME	Components of compound field: **If** numeric SELF_DELTA **Else** As-is	
FLAG	As-is	TEXT: Singleton snips
POS	POS algorithm as described above	UINT32: POS values if too distant from previous POS for delta
MAPQ	As-is	TEXT: Singleton snips
CIGAR	As-is	TEXT: Singleton snips
PNEXT	OTHER_DELTA POS *delta*	UINT32: PNEXT value if too distant from POS for delta
TLEN	**if** a non-zero value that is the negative of the previous line: “Δ -” **Else if** tlen>0 and pnext_pos_delta>0 and seq_len>0: *SPECIAL (*TLEN, tlen - pnext_pos_delta - seq_len) **Else**: As is	TEXT: Singleton snips
OPTIONAL	Container—one item per SAM tag	TEXT: Singleton snips
SQBITMAP	-	BITMAP: 0 if the base should be taken from the reference, 1 if it should be taken from NONREF (used for SEQ and E2:Z)
NONREF	-	TEXT: Based that differ from the reference (used for SEQ and E2:Z)

GPOS	-	UINT32: Position within the reference (used for SEQ and E2:Z)
STRAND	-	BITMAP: 1 if forward, 0 if reverse complement (used for SEQ and E2:Z)
QUAL	-	TEXT: QUAL data *or* DOMQUAL: data (used for QUAL and U2:Z)
QDOMRUNS	-	UINT8: Dom run lengths if DomQual algorithm is used (used for QUAL and U2:Z)
SA:Z	Container—array of: (@RNAME, @POS, @STRAND, CIGAR, @MAPQ and NM:i)	TEXT: Singleton snips
OA:Z	Container—array of: (@RNAME, @POS @STRAND, CIGAR, @MAPQ and NM:i)	TEXT: Singleton snips
XA:Z	Container—array of: (@RNAME, @POS, @STRAND, CIGAR and NM:i)	TEXT: Singleton snips
@RNAME	As-is (a subfield of SA/OA/XA)	TEXT: Singleton snips
@POS	-	UINT32: numeric values (a subfield of SA/OA/XA)
@STRAND	As-is for SA,OA ; sign of POS for XA	TEXT: Singleton snips
@MAPQ	As-is (a subfield of SA/OA)	TEXT: Singleton snips
NM:i	As-is—shared between NM:i tag and NM values within SA,OA,XA	TEXT: Singleton snips
MD:Z	**If** seq_len implied by MD:Z is identical to seq_len implied by CIGAR: *SPECIAL (MD*, value where final number is replaced with '*'). For example: `119C31` → `119C*`. In many cases there will be just a number that equals the seq_len which will be replaced '*', thereby making a highly compressible *b250*. **Else**: As-is	TEXT: Singleton snips
BD:Z BI:Z	**If** the length is equal to seq_len: The data is stored in *local* of BD_BI context, with each two bytes representing one byte from BD and one byte which is the delta between the byte from BD and the corresponding byte from BI. The BD and BI Snips themselves are: *SPECIAL* (BD_BI)	BI data (either delta vs BD or As-is)

	Else: As-is	
AS:i	***If*** *seq_len >= value* *SPECIAL (AS, seq_len - value)* ***Else*** As is	TEXT: Singleton snips
Numeric array tags *:B	Container—one item, repeats=array_len	TEXT: Singleton snips
All other SAM tags	As-is	TEXT: Singleton snips
EOL	As-is End of line—either ‘\n’ (Unix-style) or ‘\r\n’ (Windows-style)	TEXT: Singleton snips
VCF		
CHROM	As-is	-
POS	POS algorithm as described above. Used for both the POS field and the INFO/END tag	UINT32: POS values if too distant from previous POS for delta
ID	ID algorithm as described above	UINT32: numeric component of ID
REFALT	**If** `--reference` / `--REFERENCE` is used: **If** REF = reference value, set REF to ‘-’ **If** ALT = reference value, set ALT to ‘-’ **If** ALT is a single base (i.e. SNP) that is the common ALT of REF (A↔G; C↔T), set ALT to ‘+’ **If** either ALT or REF are ‘+’ or ‘-’ *Special (REFALT, REF, ALT)* **Else** *As-is*	TEXT: Singleton snips
QUAL	As-is	TEXT: Singleton snips
FILTER	As-is	TEXT: Singleton snips
INFO	Prefix followed by Container: Prefix is the INFO string including the tag names, ‘=’ and ‘;’ but excluding the values Container contains an item (context) for each INFO tag which has a value (repeats=1) Example: “AC=1;AN=2;MYFILTER” “AC=;AN=;MYFILTER *Container* (repeats=1, (AC,’’), (AN,’’))”	TEXT: Singleton snips

INFO/SVLEN	**If** a SVLEN is negative and equal to POS-END
	    Special (SVLEN)
	Else
	    As-is
		TEXT: Singleton snips

INFO/AC	**If** AC = AN * AF
	   Special (AC)
	Else
	   As-is
		TEXT: Singleton snips

INFO/END	Alias of POS	
All other INFO tags	As-is	TEXT: Singleton snips
FORMAT	As-is	TEXT: Singleton snips
FORMAT/GT	Container with one item, and repeats=ploidy, with item separator being the phase character (‘/’ or ‘|’). The haplotype data is stored in *GT.local*, as a matrix of lines x haplotypes. The matrix padded as needed to the maximum ploidy in this vblock, or in case of missing samples or lines in the vblock without GT. This matrix is then compressed at the compression stage with a *specific* codec, either *hapmat* (**REF**) or *gtshark* (**REF)**	

FORMAT/DP	**If** equal to INFO/DP
	   OTHER_DELTA (INFO/DP, 0)
	Else
	  As-is
		-

FORMAT/ MIN_DP	*OTHER_DELTA* (FORMAT/DP, DP-MIN_DP)	-
FORMAT/GL	**If** largest probability value can be calculated from the other values: Remove the largest probability value **Else** As-is	-
All other sample subfields	As-is	There is no *b250* or *local* sections for sample data; *b250* data is stored for all subfields together in a special *genotype* section as described in REF
EOL	As-is End of line—either ‘\n’ (Unix-style) or ‘\r\n’ (Windows-style)	TEXT: Singleton snips

GVF		
SEQID	As-is	-
SOURCE	As-is	TEXT: Singleton snips
TYPE	As-is	TEXT: Singleton snips
START	POS algorithm as described above.	UINT32: START values if too distant from previous START for delta
END	POS algorithm as described above, with *OTHER_DELTA* vs START	UINT32: END values if too distant from START for delta
SCORE	As-is	TEXT: Singleton snips
STRAND	As-is	TEXT: Singleton snips
PHASE	As-is	TEXT: Singleton snips
ATTRS	Same as VCF INFO	TEXT: Singleton snips
ATTRS/ID	POS algorithm	UINT32: ID values if too distant from START for delta
ATTRS/ Dbxref	ID algorithm	UINT32: numeric component of ID
ATTRS/Variant_effect	*Container* with repeats as appears in the value, and 4 items: V0arEff, V1arEff, V2arEff, ENSTid	TEXT: Singleton snips
ATTRS/sift_prediction	*Container* with repeats as appears in the value, and 4 items: S0iftPr, S1iftPr, S2iftPr, ENSTid	TEXT: Singleton snips
ATTRS/ polyphen_prediction	*Container* with repeats as appears in the value, and 4 items: P0olyPhp, P1olyPhp, P2olyPhp, ENSTid	TEXT: Singleton snips
ATTRS/ variant_peptide	*Container* with repeats as appears in the value, and 3 items: V0arPep, V1arPep, ENSTid	TEXT: Singleton snips
ENSTid	ID algorithm	UINT32: numeric component of ID
V?arEff S?iftPr P?olyPhp V?arPep	As-is	TEXT: Singleton snips
ATTRS/Reference_seq	As-is (also used to store Variant_seq and ancestral_allele)	TEXT: Singleton snips

ATTRS/ Variant_seq	Alias of ATTRS/Reference_seq	
ATTRS/ancestral_allele	Alias of ATTRS/Reference_seq	
All other ATTRS tags	As-is	TEXT: Singleton snips
EOL	As-is End of line—either ‘\n’ (Unix-style) or ‘\r\n’ (Windows-style)	TEXT: Singleton snips
23andMe		
CHROM	As-is	-
POS	POS algorithm as described above.	UINT32: POS values if too distant from previous POS for delta
ID	ID algorithm	UINT32: numeric component of ID
GENOTYPE	-	TEXT: 2 characters per genotype
EOL	As-is End of line—either ‘\n’ (Unix-style) or ‘\r\n’ (Windows-style)	TEXT: Singleton snips
FASTQ		
CONTIG	As-is	-
DESC	Container: Compound field	TEXT: Singleton snips
D?ESC	Components of compound field: **If** numeric SELF_DELTA **Else** As-is	
E1L E2L E3L E4L	As-is End of line for each one of the 4 txt lines that make up a FASTQ logical line	TEXT: Singleton snips
SQBITMAP	-	Same as in SAM
NONREF	-	Same as in SAM
GPOS	-	Same as in SAM
STRAND	-	Same as in SAM
QUAL	-	Same as in SAM

QDOMRUNS	-	Same as in SAM
FASTA		
CONTIG	As-is (first component of description)	-
DESC	Container: Compound field	TEXT: Singleton snips
D?ESC	Components of compound field: **If** numeric SELF_DELTA **Else** As-is	
LINEMETA	*Special snip* containing instructions on how to reconstruct a contig or part of a contig in this vblock	TEXT: Singleton snips
SEQ	-	TEXT: sequence
COMMENT	-	TEXT: comment lines
GENERIC		
DATA	-	All data
PHYLIP		
ID	-	SEQUENCE: ID data
SEQ	-	SEQUENCE: SEQ data
EOL	As-is End of line (Unix / Windows)	TEXT: Singleton snips

SI.3. Optimisations

Genozip, by default, is strictly lossless. However, it also offers optimisations that modify the data—modifications that are designed to be harmless for typical downstream analysis but significantly improve the compression ratio. The user may activate all optimisations with `--optimise` (or `--optimize` or `-9`) or alternatively, only specific optimisations with their respective command line options listed in Tables S5.

Table S5: Optimisations. Options that can be used with `genozip` that modify the data to make it more compressible. `--optimise` (or `--optimize`) combines all these options

Command line	File types	Algorithm
`--optimize-sort`	VCF GVF	INFO (VCF) and ATTRS (GVF): Within each line, tags are sorted alphabetically
`--optimize-PL`	VCF	PL: Phred values of over 60 are changed to 60
`--optimize-GP`	VCF	GP: Numbers are rounded to 2 significant digits
`--optimize-VQSLOD`	VCF	VQSLOD: Rounded to 2 significant digits
`--optimize-QUAL`	SAM FASTQ	QUAL (SAM, FASTQ) and U2:Z (SAM): quality phred scores are binned, an similar to Illumina binning, but extended
`--optimize-ZM`	SAM	ZM:B: Negatives are changed to zero, and positives are rounded to the nearest 10
`--optimize-DESC`	FASTQ	Replaces the description line with '@filename:read_number'
`--optimize-Vf`	GVF	Variant_freq: Rounded to 2 significant digits

SI.4. Compression against a reference and the Genozip Aligner

Overview

Genozip provides the ability to compress against a reference genome, in four cases:

a. FASTQ files
b. Unaligned SAM files
c. Aligned SAM files
d. VCF files (REF and ALT fields)

The reference is used in two distinct ways: in cases c and d, the file contains the position in the reference file, and we simply compare the file data to the reference data at the position provided. In cases a and b, the file does not contain positional information regarding the location of a particular read, and we use the Genozip Aligner to generate this position. In some cases where SAM files contain lines with and without POS information, we may compress the lines relying on the POS information where it is provided, and use the Genozip Aligner where it is not.

Most (if not all) aligners currently available have the objective of finding the true location an actual DNA fragment had in the original DNA molecule prior to the sequencing process. In contrast, the Genozip Aligner doesn't attempt to find the true location of a read, all we need to find is a location in the reference file that is significantly similar to the read so that we can use this similarity for better compression. This subtle difference in objective allows us to create an algorithm that is radically different from traditional aligners, trading off positional accuracy for speed.

The algorithm is divided into two:

1. Processing of a FASTA file into a reference file with `genozip --make-reference`. This step needs to be run only once, and the resulting reference file may be used to compress subsequent data files of the same species. The resulting reference file, distinguished by an extension `.ref.genozip`, contains mostly sections of two types REFERENCE and REF_HASH which shall be described below.

2. Compressing a data file against a reference file with either `genozip --reference` or `genozip --REFERENCE`. In the former the reference file needs to be provided to

`genounzip` when decompressing, while in the latter the needed parts of the reference are stored as part of the compressed file, so the reference file is not needed for `genounzip`. This is particularly useful when binding together (i.e. compressing into a single `genozip` file) multiple files for delivering to a customer, as the cost in file size of storing the reference is amortised across multiple data files, and the customer doesn't need to worry about dealing with a reference file.

The REFERENCE data

When generating a reference file with `genozip --make-reference` *`fasta-file.fa`* a REFERENCE section is outputted for each *vblock* of FASTA data processed. The division of the fasta file into vblocks is constrained so that each *vblock* contains data from only a single contig—possibly the whole contig if it is short enough to fit in a *vblock*.

The REFERENCE data is simply a 2-bit representation of the FASTA data, where 'A' and 'a' are represented by 00, 'C' and 'c' by '10' (1) ; 'G' and 'g' by '01' (2) ; 'T' and 't' by '11' (3). Any other character contained in the FASTA data, including 'N', is represented by 00 as well. Every four characters are fit into a 8-bit byte, and the section is further compressed with *lzma*.

Note regarding bit notation: throughout Genozip, we store bits in bit arrays. These bit arrays are made out of 64 bit words, which is the native word size in most modern CPUs. When describing these bit arrays in this paper, we do so in two equivalent ways:

1. Little Endian: thinking of the bit array as a string of bits corresponding to a string of nucleotides—we write it e.g. 'ACG' as '001001'—enclosed in a single quote.
2. Big Endian: we can also think of the bit array as a binary number. Consistent with the normal way of describing numbers, we start with the most significant bit(without quotes, and prefixed with 0b) 0b100100 in this example, which is 36 in decimal.

We also store for each contig its GPOS (short for Global Position), a 32-bit unsigned integer that starts from 0 for the first contig, and is set for each subsequent contig to be higher than the (GPOS + length) of the previous contig.

The Genozip Aligner uses GPOS for describing the position of reads in the reference rather than (contig, pos).

We chose to store GPOS in an unsigned 32-bit integer, thereby limiting our Genozip Aligner to the first 4 Gbp of a reference. This is sufficient for single-species references commonly used today. For example, GRCh38 contains about 3.2 Gbp.

In the future, we might want to support references larger than 4 Gbp, in particular multi-species references that might be useful in metagenomics, which will require GPOS to be longer than 32 bits. Genozip already treats all position data (POS and GPOS) as 64-bit integers internally, so this could be relatively easily supported, but with the cost being achieving slightly worse compression ratios as GPOS data is also contained in `genozip` files of data files compressed using the Genozip Aligner.

The REF_HASH data

The second big chunk of data generated when generating a reference file with `genozip --make-reference` *`fasta-file.fa`* is REF_HASH data. This is a pyramid of ***num_layers*=**4 hash tables, of levels [0, 3], where each hash table contains $2^{28-\textit{layer}}$ entries. Each entry is a 32-bit unsigned integer, which will contain a particular GPOS, or remains at the initial value 0xffffffff if not used.

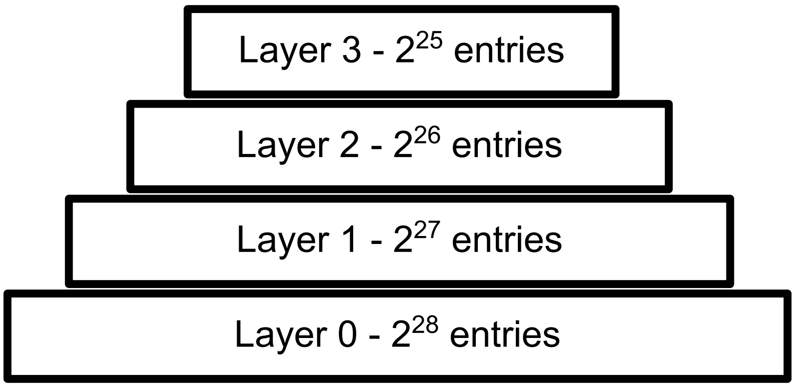

Layer 3 - 2^{25} entries
Layer 2 - 2^{26} entries
Layer 1 - 2^{27} entries
Layer 0 - 2^{28} entries

Figure S2: ref_hash tables. Layer 0 is attempted first, and if it is occupied, progressively higher layers are utilised

The ref_hash tables are created by a single traversal of the reference data described above, from ***gpos***=0 to the last GPOS value:

```
create_ref_hash:
    Initialise all ref_hash table entries to 0xffffffff
    Foreach locus gpos in reference which is 11 (i.e. 'G' or 'g') {
        If base at (gpos+1) is not also 11 {
            Let idx ← value of the next 28 bits (i.e. 14 bases) following the G
            Insert (idx, gpos)
        }
    }

Insert (idx, gpos):
    If      (ref_hash[0][idx] == 0xffffffff) then ref_hash[layer][idx]  ← gpos
    Else If (ref_hash[1][idx] == 0xffffffff) then ref_hash[layer][idx]  ← gpos
    Else If (ref_hash[2][idx] == 0xffffffff) then ref_hash[layer][idx]  ← gpos
    Else If (ref_hash[3][idx] == 0xffffffff) then ref_hash[layer][idx]  ← gpos
    Else if (random chance of 25%)
        ref_hash[random(0 to 3)][idx] ← gpos
```

Notes:

- The notation ref_hash[***layer***][***idx***] means the ref_hash table at layer ***layer***, at the index ***idx*** where only the needed least significant bits of idx are used (28 bits for level 0 down to 25 bits for level 3).

The resulting 4 ref_hash tables are compressed with *lzma* and written to the genozip reference file.

Compressing aligned SAM data

When compressing aligned SAM data (i.e. a SAM line for which we have the RNAME and POS), we use the ***reference*** data in the reference file, however we don't need the ***ref_hash*** tables and the GPOS data.

We process the data into the following data structures:

- SQ_BITMAP context: a bit array for which we have 1 bit for each base in the sequence which according to the line's CIGAR string consumes both Query and Reference as defined in the SAM spec **REF** page 8. The bit will be one if the decompressor should copy this base from the ***reference*** and 0 if it should get it from NONREF.

- NONREF context: a character array that stores all the bases that are different from the reference or that are not one of 'A','C','G','T'.

- REF_IS_SET: a bit array that contains one bit for each base (2 bits) in ***reference***. The bit is 1 if and only if the value of ***reference*** at this location will be needed for reconstructing the data during `genounzip`. It is used only in case of --REFERENCE (i.e. not --reference), to determine which parts of the reference should be written to the file.

The SAM segmenter, when segmenting a particular line within a *vblock* of SAM data, traverses the SEQ data according to the CIGAR data:

```
Foreach base in SEQ:
    Let cigar be the Op in CIGAR string covering base (as defined in REF):
        If cigar ∈ { 'M', '=', 'X' }
            If (base == reference[RNAME,POS(base)])
                SQ_BITMAP ← 1
                REF_IS_SET[RNAME,POS(base)] ← 1
            Else
                SQ_BITMAP ← 0
                NONREF ← base
    Else if cigar ∈ { 'I', 'S' }
        NONREF ← base
```

Notes:

- *POS*(*base*) is the POS value of base calculated from the POS value in the line, and the CIGAR string, relative to the position of the base in the SEQ string
- Assignment to SQ_BITMAP and NONREF means adding one value at the end of the array

Compressing VCF data (REF and ALT fields) with a reference

When compressed without a reference, we store the REF and ALT data together (separated by a tab character) in the REFALT context. We do this as they are obviously correlated and hence storing them together results in better compression than storing them in separate contexts.

We the user specifies a ***reference*** (using `--reference` or `--REFERENCE`) we do the modify the string stored in the context in the following way:

If REF or ALT is the same base as in (CHROM,POS) in ***reference***, we store '-' instead of the base.

If ALT is the *common_snp*(REF), then we store '+' instead of the base.

Notes:

- *common_snp:* A↔G ; C↔T

- Since we don't force the user to use the same reference for compressing as used for generating the VCF, we can't assume the value of the REF field is the same as in ***reference***. However, we expect this to be the usual case. In this case, all the REF values will be replaced with '-'

- We also set REF_IS_SET exactly as described above for aligned SAM compression

The effect of this is to make the b250 data of the REFALT context a lot more compressible due to both the reduction of the dictionary size as well as the abundance of the "-\t+" word.

The significance of this algorithm on the overall VCF compression ratio depends on the significance of the REFALT contribution to the file's information content. In files that don't contain any samples or INFO data, the REFALT data tends to be a major contributor to the information content, while in files that are rich in INFO data and contain many samples with multiple subfields, the contribution of REFALT information to the file's information content will be minor.

Compression of FASTQ sequence data and SEQ fields in unaligned lines in SAM data

For a read sequence data (which we will refer as SEQ hereinafter) for which we don't have alignment information, as is the case in FASTQ and unaligned lines of a SAM file, we use the REF_HASH data to find the a ***gpos*** value which represents the beginning of a region of the ***reference*** that we choose to compress against. We attempt to find a ***reference*** region identical to the sequence at hand, or with a small amount difference.

When loading the ***reference*** data from the reference file, we store two copies of it in memory—one forward copy and one reverse complement copy.

Notes:

- We do not attempt to find the region with the absolutely smallest amount of difference, however this is very often the outcome nevertheless, in particular in the common case where there is only a single region in the genome with which the SEQ aligns reasonably well.

- We do not handle insertions and deletions (Indels), resulting in reads which contain Indels typically aligning well only to the longest sub-read segregated by Indels. However, since typically insertion and deletions appear only in a small percentage of the reads, this has minimal effect on the compression ratio.

- When comparing sequences to the ***reference***, we compare to both the forward and the reverse complement references. The ***gpos*** value stored is the lowest gpos of a base, which is the first base in case SEQ aligns to the forward reference, and the last base in case it aligns to the reverse complement.

- In SAM, It is possible that a *vblock* contains both aligned and unaligned reads. In this case, we will compress each read with the appropriate algorithm, resulting in the SQ_BITMAP and NONREF contexts containing data used to compress both types of lines.

- Since the Genozip Aligner algorithm does not handle Indels, it will not generate good compression ratios for reads created using long read sequencing technologies, such as PacBio SMRT or Oxford Nanopore, that are rich in erroneous Indels. These files will compress better if the data is first aligned into a SAM/BAM file using an appropriate aligner, and then compressed as an aligned SAM.

We process the data into the following data structures:

- SQ_BITMAP context: a bit array for which we have 1 bit for each base in the sequence at hand. The bit will be one if the decompressor should copy this base from the ***reference*** and 0 if it should get it from NONREF. Note that unlike compressing aligned SAM reads described above, here we have no CIGAR.

- NONREF context: a character array that stores all the bases that are different from the reference or that are not one of ‘A’,’C’,’G’,’T’.

- GPOS context: the context *local* data stores unsigned int GPOS values, one for each read. This is the lowest GPOS value of the sequence at hand as explained above.

- STRAND context: the context *local* data stores a bitmap, one bit for each read, which is 1 if this read is to be reconstructed against the forward reference and 0 if it is to be reconstructed against the reverse complement reference.

- REF_IS_SET: used exactly as described for aligned SAM compression.

```
align (SEQ):
    Foreach base in SEQ:

        If base is a ‘G’ and the preceding base is not a ‘G’:
            score, gpos = score_match (SEQ, location of ‘G’ in SEQ, ‘forward’)
            If score is the highest so far for SEQ
                best_match ← score, gpos, ‘forward’

        If base is a ‘C’ and the next base is not a ‘C’”
            score, gpos = score_match (SEQ, location of ‘G’ in SEQ, ‘reverse’)
            If score is the highest so far for SEQ
                best_match ← score, gpos, ‘reverse-complement’

score_match (seq, ‘G’ location, strand):
    idx ← numeric_value(14 nucleotides following the G)
```

gpos ← ref_hash[***idx***]

ref ← copy of the region of ***reference*** *or* ***reverse-complement-reference*** (determined by ***strand***) which is aligned to ***seq*** according to ***gpos***

Implementation details: at this point ***seq*** and ***ref*** are two bit arrays of identical length. Every two bits represent a nucleotide (0=A,1=C,2=G,3=T). The arrays are implemented as 64-bit words, so that each 64 bit word contains (up to) 32 nucleotides, starting at the first bit of the first word, and with the redundant bits of the last word set to zero.

score ← length (***seq***) - *count_1_bits* (***seq*** bitwise-XOR ***ref***)

Note: *count_1_bits* counts the number of bits that are '1' in a bit array

<u>Example</u>:

SEQ="CACTCT**G***TTCGCAGCAGTCTG*CGCCCTTACACAAAATG"

Consider the 14 nucleotides following the G: for example:
"CACTCT**G***TTCGCAGCAGTCTG*CGCCCTTACACAAAATG"
Or, the 28 bits representing the same 14 nucleotides:
'1111100110000110000111101101' which is numerically
0b1011011110000110000110011111 (in binary) or 192,438,687 (in decimal).

gpos ← refhash[192,438,687] = 500000000 (example)
500000000 in this example is the coordinate in the whole-genome reference of the **G**
Consider the reference segment around this G, so that it is aligned to SEQ:

```
            SNP                                     SNP         DEL
REF:"CATTCTGTTCGCAGCAGTCTGCGCCCTTTCACAAAGAT"
```

Using 64-bit words, bitwise-XOR the reference segment and SEQ

```
SEQ:'1000101110110111111001100001100001111011011001101010111100100010 000000001101'
REF:'1000111111011011111100110000110000111101101100110101011111100010 000000010011'
XOR:'0000010000000000000000000000000000000000000000000000000011000000 00000011111'
       WORD #1                                                         WORD #2
```

score = seq_len_in_bits - count_1_bits(XOR) = 74 - 8 = 66

Alternative contig names

When compressing a file using a reference, if a contig name that appears in the file does not appear in the reference, we attempt to search for it using alternative names:

- A number eg “22” is also searched with a “chr” prefix, “chr22”
- “M” and “chrM” are also searched as “chrMT”

The alternative contig names are searched and assigned during compression, and the mapping is stored in the genozip file in the `SEC_ALT_CHROMS` section.

Discussion

The algorithm scores candidate matches (alignments) by doing bitwise operations on entire 64-bit words containing 32 bases each. The entire scoring takes only a few CPU operations per 64-bit word, as little as 16 (depending on the CPU). This is the key component that makes the Genozip Aligner extremely fast.

Note that rather than counting the number of mismatching bases, we count the number of mismatching bits, and select the “match” (i.e. the reference locus) with the least mismatching bits. This is done with a single CPU instruction per 64-bit word on most modern CPUs (for example, `popcntq` on Intel CPUs). As a result, we don’t necessarily select the match with the least mismatching bases. However, in practice, the majority of reads will have exactly one locus which provides a good match which will be selected, and in the case where we have two or more loci that are all good matches (i.e., very low mismatching bases) and the one we select, with the lowest mismatching bits, is not the one with the lowest mismatching bases; this minute difference would be insignificant for compression purposes.

Sensitivity to Indels: since we compare whole bit arrays, we don’t handle Indels. This usually means that the match selected for a read that contains an Indel would be the one matching the largest sub-read, as segregated by Indels. For typical short-read data, the percentage of reads containing Indels is sufficiently small that this approach results in very good compression ratios. However, long read technologies available at this time often generate reads that are enriched in Indels that are not correct biologically but are rather artifact of limitations of these technologies. This algorithm does not work well with this type of long read data, and we advise users to first align the data using an appropriate aligner, and then compress the resulting SAM (or BAM) file, which would use the aligned SAM algorithm described earlier.

Computational complexity: Since the number of loci we test is proportional to the length of the read (we test all ‘G’ bases for forward matching and all ‘C’ bases for reverse complement matching), and each test compares the entire read to the candidate region in the reference; the complexity of aligning a SAM or FASTQ file is $O(r^2 n)$ with r being the read length and n being the number of reads.

Optimisation: if the command line option `--fast` is specified, we test every 5th base to see if it is a ‘G’ or ‘C’ rather than every base. On our test data, this resulted in a speed up of the alignment by about 4X at the cost of a 20% worse compression ratio.

SI.5. Compression of FASTQ paired end read files

Genozip provides a command line option `--pair` that further optimises compression in case of paired-end FASTQ files, such as Illumina. When using this option, every two consecutive input files on the command line are assumed to be a pair.

This optimisation consists of:

1. For each component of the Description (i.e. the D?ESC contexts): if the value of the second paired file is identical to the first, store the snip LOOKUP_PAIR instead of the value. For most D?ESC contexts (all but one in Illumina files), the entire *b250* data would be a run of LOOKUP_PAIR compressing to a trivial size.

2. For GPOS in the second paired file: we store it as a PAIR_DELTA snip vs the GPOS of the first file.

3. For STRAND (in the second pair file): we store ‘1’ if the second paired read direction (i.e. forward or reverse complement) is identical to that of the first paired read, and 0 otherwise. This is expected to result in long, highly-compressible runs.

SI.6. Specific codecs

***acgt*: A *specific* codec for compression of nucleotide sequences**

We observe that *lzma* compresses nucleotide sequence data well, however it is very slow. *acgt* is a new codec we present here, specifically for nucleotide sequences, which is about 25X faster on our test data at a cost of only 5% worse compression than *lzma*. We use *acgt* for compressing NONREF data in FASTQ as well as aligned and unaligned SAM, unless the user specifies `--optimize-SEQ` or `--optimize` in the command line, in which case we compress NONREF with *lzma*.

acgt compression is designed for nucleotide sequence data in which ‘A’, ‘C’, ‘G’ and ‘T’ characters make up the vast majority of the data, while other characters, such as ‘N’, are rare. The source data is expected to be textual ASCII data.

acgt splits the data into two streams, each which is outputted to a separate section in the genozip output file.

The first stream, CODEC_ACGT, is simply the sequence encoded in 2-bit as we do in REFERENCE explained above: ‘A’/’a’ are encoded as 0b00, ‘C’/’c’ as 0b01, ‘G’/’g’ as 0b10, ‘T’/’t’ as 0b11 and everything else as 0b00. The size of the data is ¼ of the original size since we’re converting each byte into 2-bits. We then further compress the CODEC_ACGT with *lzma*.

The second stream, CODEC_XCGT, is used only if we have one or more characters that are not ‘A’, ‘C’, ‘G’ or ‘T’. This is an array of bytes of the same length as the source data. We set it to 0 for every character that is ‘A’, ‘C’, ‘G’ or ‘T’ in the source data, 1 for every character that is ‘a’, ‘c’, ‘g’ or ‘t’ (i.e. lowercase) and we leave other characters as they are in the source data for all other characters. We then compress CODEC_XCGT with *bz2*. Since the *bz2* algorithm contains run-length encoding, and since we expect the vast majority of characters to be 0, this compresses to a very small size in real world cases we tested.

***hapmat*: A *specific* codec for compression of a haplotype matrix**

For FORMAT/GT data in VCF files, we use the algorithm described in REF to compress a matrix, who's lines represent variants, columns represent haplotypes (here, we loosely use the term *haplotype* to describe the column of a specific sample at a specific location in the GT value, even if the sample is not phased), and each entry in the matrix is a single character representing the allele. genozip supports alleles up to 99, where alleles 10 and above are rewritten as a single ASCII character. This algorithm has been now implemented as a codec:

- For each haplotype column, count the number of alternate alleles (allele 1 to 99)
- Sort the haplotype columns by the count alternate alleles
- Transpose the sorted matrix
- Compress the transposed matrix with bzip2

The results of this compression are stored in two contexts:
GT_HT.local stores the compressed matrix
GT_HT_INDEX.local stores the permutation index that describes how to un-sort the matrix back to its origin.

We note that there are better algorithms for compression of a haplotype matrix based on Positional Burrows Wheeler Transform, as described in (**REF**), and this might be an area for improvement in the future.

***DomQual*: a specific codec for compression of base quality scores**

Compression of sequences of base quality scores, as they appear in FASTQ files and in the QUAL field of SAM files, are often a harder problem than that of nucleotide sequences, because there is no reference data to which we can compare base quality scores. Further, different sequencing technologies and even different versions or options selected within the same sequencing technology, generate quality scores with radically different patterns.

Here, we introduce a novel algorithm to address a specific pattern of quality scores that is very common. This pattern is defined by having a single quality score that dominates the sequence.

In Genozip, we decide for each particular *vblock* of FASTQ or SAM data whether to use *DomQual* by sampling the first 500 quality scores (i.e. characters) of the base quality data of each of the first 5 lines of the *vblock* (in FASTQ a *vblock* line means a 4 textual lines of the FASTQ file). If there is a single character that accounts for at least 50% of the number of characters sampled, then we use *DomQual* for the base quality data in this *vblock*, and set ***dom*** to the dominant character of the sample.

In real-world data we tested, *DomQual* will usually triggered in Illumina files with quality binning (**REF**) where the dominant character is usually ‘F’, Pac Bio CCS data where the dominant character is usually ‘~’, and quality data that has been binned with the genozip option `--optimize-QUAL` or `--optimize`.

We consider the entire base quality data of the vblock as a single long sequence of quality scores. *DomQual* segments this sequence data into the *local* data of two contexts:

- QUAL context contains a copy of the sequence, with two changes:

1. All ***dom*** characters removed
2. For each remaining character (which by definition is a non-***dom***): if this character is NOT preceded in the source sequence by a *dom run* (which we hereby define as one or more consecutive ***dom*** characters), a byte with the value of 1 is inserted before this non-***dom*** in QUAL.

- QDOMRUNS context contains a length of each *dom run*, the sub-sequence of the source sequence containing one or more ***dom*** characters, preceding each non-***dom*** character, except those we marked with 1. These are represented by a single byte indicating the length (between 1 and 254). If the length of the *dom run* is more than 254, then we add 1 or more 0xff characters each, representing a length of 254. Example: 0xff 0xff 0x08 indicates a *dom run* of length 254 + 254 + 8 = 516.

The *local* data of both the QUAL and QDOMRUNS contexts is compressed with *lzma*.

Testing with Illumina binned quality data, we see that *bz2* achieves superior compression to *lzma*, and is faster than it. On our test data, the *DomQual* codec achieves 12% better compression than *bz2*, and is slower than *bz2* by a factor of 3.5X.

If the user specified the command line option `--fast`, we compress these contexts with *bz2* instead of *lzma*. On our test data, this resulted in a compression that is about 3% better and about 10% faster than *bz2* on the source quality sequence.

SI.7. Random access, subsetting & pipeline integration

Genozip contains capabilities to allow genozip files to be directly integrated in analysis pipelines, as well as some internal subsetting capabilities.

`genozip` supports reading and writing txt files from a pipe, for example:

```
cat myfile.fq | genozip - --output myfile.fq.genozip
```

```
genocat myfile.fq.genozip | analysistool
```

Some analysis tools require random access to the txt file and hence cannot accept an input file on a pipe. In these cases, it would be necessary to genounzip the file first.

`genocat` is a tool for viewing the data within genozip file, and potentially subsetting it. The subsetting command line options are summarised in Table S6, more details are available by running `genocat --help`.

Random access (`--regions`) is implemented by a two global sections in the genozip file:

1. The `SEC_RANDOM_ACCESS` section is included in all genozip files of file formats on which `--regions` is supported. It contains an array for a record for each *vblock* (the list of contigs appearing in the *vblock)*, and for each contig, the first and last position within the contig appearing in the *vblock.* The contents of this section may be viewed using the `--show-index` command line option.

2. The `SEC_REF_RAND_ACC` section is included in reference files, and also in genozip files that are compressed with `--REFERENCE`. contains a similar array, but with a record for each `SEC_REFERENCE` section.The contents of this section may be viewed using the `--show-ref-index` command line option.

When using `genocat --regions`, genozip uses the information from these two sections to refrain from reading from disk *vblocks,* `SEC_DICT` sections and `SEC_REFERENCE` sections that contain no data from the requested regions.

Table S6: genocat options. A partial list of the options of `genocat` - those options that subset the file. See `genocat --help` for a full list of options.

`genocat` option	**File formats**	**Action**
`--downsample <rate>`	All	Include only one line (or read for FASTQ) per *rate* lines.
`--regions <`*`region-list`*`>`	VCF, SAM, FASTA, GVF, 23andMe, reference file	Include or exclude specific contigs and/or positions
`--samples <`*`sample-list`*`>`	VCF	Include or exclude specific samples
`--grep <`*`string`*`>`	FASTQ, FASTA	Show only reads (FASTQ) or contigs (FASTA) whose describe contains the *string*
`--drop-genotypes`	VCF	Exclude the FORMAT and samples columns
`--no-header`	All	Exclude the header lines
`--header-only`	All	Include only the header lines
`--header-one`	VCF, FASTA	In VCF, includes only the last of the header lines (with the field sane sample names). In FASTA, includes only the first component of the description line (until the first space).
`--GT-only`	VCF	Exclude all sample subfields, except for GT
`--sequential`	FASTA	Output the sequence of each contig as a single line, removing any newlines
`--list-chroms`	VCF, SAM, FASTA, GVF, 23andMe, reference file	List the names of the chromosomes (contigs)

SI.8. Tools for obtaining statistics and metadata

Genozip contains tools for obtaining additional information about the contents of files. These are summarized below. More details can be obtained by running `genozip --help -f`.

Table S7: Statistics and metadata options, provide deep insight into the data in the files being processes, as well as the execution flow of the Genozip algorithms

Option	**Availability:** **Z** **`genozip`** **U** **`genounzip`** **C** **`genocat`** **L** **`genols`**	**Action**
`--show-time`	ZUCL	Show profiling information of where execution time was spent
`--show-memory`	ZUCL	Show memory consumption information
`--show-stats`	Z	Show compression performance by context
`--SHOW-STATS`	Z	Show detailed context information
`--show-alleles`	Z	(VCF only) show alleles
`--show-dict`	ZUC	Show all dictionary fragments
`--show-one-dict` *`<context>`*	ZUC	Show dictionary fragments of *context*
`--list-chroms`	ZUC	List the names of the chromosomes (contigs)
`--show-gt-nodes`	Z	(VCF only) show the GT values matrix (transposed)
`--show-b250`	ZUC	Show contents of all *b250* sections (textual)
`--show-one-b250` *`<context>`*	ZU	Show contents of one *b250* section (textual)
`--dump-one-b250` *`<context>`*	ZUC	Dump the contents of a *b250* as it appears in the file (binary)
`--dump-one-local` *`<context>`*	ZUC	Dump the contents of a *local* as it appears in the file (binary)
`--show-headers`	ZUC	Show a subset of the contents of the genozip file section headers as they are read or written
`--show-index`	ZUC	Show the contents of the `SEC_RANDOM_ACCESS` section

Continued from previous page

Option	**Availability:** **Z** `genozip` **U** `genounzip` **C** `genocat` **L** `genols`	**Action**
`--show-reference`	ZUC	Show the ranges included the `SEC_REFERENCE` sections
`--show-ref-seq`	ZUC	Show the reference sequences
`--show-ref-index`	ZUC	Show the contents of the `SEC_REF_RAND_ACC` section
`--show-ref-hash`	ZUC	Show details of `SEC_REF_HASH` sections
`--show-ref-contigs`	ZUC	Show the details of the reference contigs
`--show-ref-alts`	ZUC	Show contents of `SEC_ALT_CHROMS` section
`--show-gheader`	ZUC	Show list of sections in this file, as it appears in the `SEC_GENOZIP_HEADER` section
`--show-vblocks`	ZUC	Show vblock headers as they are read / written
`--show-threads`	ZUC	Show thread dispatcher activity
`--show-hash`	Z	See the values of the parameters used for calculating the hash table size for each context
`--show-aliases`	ZUC	Show the `SEC_DICT_ID_ALIASES` section
`--debug-memory`	ZUCL	Show memory buffer allocations and destructions
`--debug-progress`	ZUC	See data related to the progress indicator
`--show-reference`	ZUC	Show details of the `SEC_REFERENCE` sections
`--show-is-set` *`<contig>`*	UC	Shows the contents of `SEC_REF_IS_SET` sections of *contig*
`--show-bgzf`	ZUC	Show details of BGZF blocks
`--show-containers`	UC	Show flow of container reconstruction
`--show-txt-contigs`	ZUC	Show contigs from the SAM/BAM header
`--show-mutex`	ZUCL	Show locks and unlocks of a particular mutex
`--show-digest`	ZUC	Show MD5 and Adler32 updates

source: the data in this table is based on the output of `genozip --help=dev`

SI.9. CPU scalability: synchronisation and thread management

Genonzip threads are managed by a thread dispatcher. The dispatcher is used both by the main I/O thread loop as described in the architecture diagram, as well as for various secondary tasks throughout the code. The dispatcher dispatches raw *vblocks* to threads, collects processed *vblocks* upon thread completion, and updates the progress indicator.

The maximum number of concurrent threads is either set by the user with the `--threads` command line option, or is set to the available number of logical cores as retrieved from the operating system.

Actually utilizing a large number of cores is a challenge, as genozip contexted-oriented compression implies that the dictionary of each context is potentially grown by every *vblock* which contributes values to a dictionary that were not observed before. When multiple threads are running in parallel each attempting to update dictionaries, the synchronisation required (for example, blocking on a mutex while updating a dictionary), if implemented naively, would severely limit the number of threads that can actually run concurrently.

In addition to the dictionaries themselves, genozip also maintains a hash table per context, which allows an efficient search when a *compute thread* is searching for a dictionary index of a particular snip. These hash tables also evolve with each snip that is added to a dictionary.

To address this, genozip *context manager* maintains, for each context, a *z_context*. When a compute thread for a specific vblock starts, the *z_context* and associated hash tables are cloned into the vblock context. This cloning doesn't actually copy memory, but rather points *z_context* and hash tables, and includes various parameters to limit this compute thread's access to only the parts of the data that were available at the point in time of the cloning—effectively creating a read-only replica of the *z_context* at this point in time, but without the expensive operation of copying memory.

As the compute thread processes the vblock, it adds new discovered snips to its own private fragments, and its own hash tables.

After the vblock processing is complete, the context manager (running in the compute thread) merges this context's data back into the zfile data. Since multiple compute threads running in parallel may have added the same snip, these merges needed to be serialised and make sure that the snips added were not already added by a previous compute thread. For this synchronisation, we use mutexes in the most sparing way possible, and opting for carefully crafted sequences of CPU-atomic operations, therefore not blocking threads, in lieu of mutexes, wherever possible.

SI.10. Security

DNA data is legally considered in many jurisdictions as “personally identifiable information” (PII) and as such is required to be secured.

Genozip provides built-in security that is easy to use.

When the `--password` is used, the genozip file is encrypted with the standard AES encryption, using the a 256-bit encryption key is generated for each section, derived from the password, the vblock number (*vb_i*) and the section type.

For padding the last block of each section to the 16-byte AES block size, a secure padding derived from the MD5 hash of the last 100 bytes of the section.

Accessing this file using `genounzip` or `genocat` is made possible only if the same password is provided.

In addition, using `genozip` with the `--md5` or `--test` options calculates the MD5 signature of the original txt file(s). Then, when using `genounzip` with `--md5`, the MD5 signature of the actual output file is compared to the MD5 stored in the genozip file, to ensure the file was not tampered with intentionally or accidentally.

Note that the MD5 calculated is that of the underlying textual file. For example, when compressing a .sam.gz or .bam file, the MD5 will be that of the underlying .sam file.

SI.11. Genozip file format

The genozip file consists of *sections*, where each section is of a particular *section type* and consists of a *header* and a *body*. The header is a structure determined by the section type, while the body contains the actual data.

All numeric data is stored in Big Endian.

Genozip supports binding multiple files together into a single genozip file. We will refer to the compressed data of these each txt files as a *component* of the genozip file.

First to appear in the genozip file, are sections related to the *component* data. Each component consists of a `SEC_TXT_HEADER` section followed by 1 or more *vblocks*. Each vblock consists of a `SEC_VB_HEADER` section and all the contexts of this vblock which include any number of `SEC_B250` and `SEC_LOCAL` sections. In the case of VCF, these may also include `SEC_VCF_GT_DATA, SEC_VCF_PHASE_DATA, SEC_VCF_HT_DATA` and `SEC_VCF_HT_GTSHARK` sections.

Following the components, we have *global sections* that apply to the entire file.

As the last section of the file, we have the `SEC_GENOZIP_HEADER` section. This contains the header, a body which is the list of sections in this genozip file and their offsets, and, unlike other sections, this section also has a *footer* which appears at the very end of the file, and contains the offset of the beginning of the section.

When a genozip file is read (for example during `genounzip`), the footer is consulted first for the the offset of the `SEC_GENOZIP_HEADER` section, then the `SEC_GENOZIP_HEADER` is read, and then the required sections from the rest of the file, based on the section offsets that are in the body of the `SEC_GENOZIP_HEADER` section.

Below is an example of the sections of a VCF file. This is a very small VCF file containing 3494 lines of the VCF header data, followed by 6 data lines, each with just a handful of INFO tags, one sample and a few sample subfields. The list below is taken from the output of running genozip with the `--show-gheader` command line option.

Real-world genozip files can often contain tens of thousands of sections.

The full list of section types and format of the header of each appear in `https://github.com/divonlan/genozip/blob/master/sections.h`

The first section is the *component* header, whose body is the VCF txt file header. This small genozip file contains only one component, but genozip supports multiple components.

```
 0. SEC_TXT_HEADER                    vb_i=0 offset=0 size=22277
```

First (`vb_i=1`) *vblock* header. This is a small VCF file that contains only one *vblock*

```
 1. SEC_VB_HEADER                     vb_i=1 offset=22277 size=93
 2. SEC_B250               CHROM      vb_i=1 offset=22370 size=46
 3. SEC_B250               POS        vb_i=1 offset=22416 size=46
 4. SEC_B250               ID         vb_i=1 offset=22462 size=46
 5. SEC_B250               REF+ALT    vb_i=1 offset=22508 size=46
 6. SEC_B250               QUAL       vb_i=1 offset=22554 size=46
 7. SEC_B250               FILTER     vb_i=1 offset=22600 size=46
 8. SEC_B250               INFO       vb_i=1 offset=22646 size=46
 9. SEC_B250               FORMAT     vb_i=1 offset=22692 size=46
10. SEC_B250               AF         vb_i=1 offset=22738 size=41
11. SEC_B250               AN         vb_i=1 offset=22779 size=41
12. SEC_LOCAL              AC         vb_i=1 offset=22820 size=41
13. SEC_VCF_GT_DATA                   vb_i=1 offset=22861 size=40
14. SEC_VCF_HT_DATA                   vb_i=1 offset=22901 size=40
```

This is the global area of the file. It starts with dictionary fragment sections—each *vblock* may or may not contribute a dictionary fragment to each context.

```
15. SEC_DICT               CHROM      vb_i=0 offset=22941 size=999
16. SEC_DICT               POS        vb_i=1 offset=23940 size=43
17. SEC_DICT               ID         vb_i=1 offset=23983 size=42
18. SEC_DICT               REF+ALT    vb_i=1 offset=24025 size=44
19. SEC_DICT               QUAL       vb_i=1 offset=24069 size=42
20. SEC_DICT               FILTER     vb_i=1 offset=24111 size=47
21. SEC_DICT               INFO       vb_i=1 offset=24158 size=133
22. SEC_DICT               FORMAT     vb_i=1 offset=24291 size=50
23. SEC_DICT               AF         vb_i=1 offset=24341 size=42
24. SEC_DICT               AN         vb_i=1 offset=24383 size=42
25. SEC_DICT               AC         vb_i=1 offset=24425 size=45
26. SEC_DICT               DP         vb_i=1 offset=24470 size=42
27. SEC_DICT               RGQ        vb_i=1 offset=24512 size=42
```

This file was compressed with `--REFERENCE` and therefore contains the relevant parts of the reference data. First, in the list of all contigs in the reference file, followed by the reference data itself.

```
     28. SEC_REF_CONTIGS                    vb_i=0 offset=24554 size=2665
     29. SEC_REFERENCE                      vb_i=1 offset=27219 size=60
```

A list of context aliases

```
     30. SEC_DICT_ID_ALIASES                vb_i=0 offset=27279 size=44
```

These are an index describing the contigs and their position range within each *vblock* allowing for efficient random access, such as when subsetting with `genocat --regions`

```
     31. SEC_RANDOM_ACCESS                  vb_i=0 offset=27323 size=52
     32. SEC_REF_RAND_ACC                   vb_i=0 offset=27375 size=52
```

The genozip header

```
     33. SEC_GENOZIP_HEADER                 vb_i=0 offset=27427 size=758
```

SI.12. Detailed results data

Compressing against raw files

We evaluated the performance of Genozip by comparing it to several widely adopted genomic data compression tools using a set of standard benchmarking files provided by the National Institute of Standards and Technology's Genome in a Bottle (GIAB) project (Table S8).

Variant Call Format (VCF)

To benchmark Genozip's VCF compression performance, we compressed the GIAB v3.3.2 NA12878 single-sample VCF and compared the results against several other popular compression tools – gzip (Meyering), pigz (Adler, 2014), BCF compression implemented in BCFtools (Li, 2011a), and bzip2 (Seward, 1996).

Sequence Alignment Map (SAM)

To benchmark Genozip's SAM compression performance, we used the 30X downsampled BAM file from GIAB that was converted to SAM format using samtools v1.9 (Li *et al.*, 2009). Genozip performance was compared to CRAM compression, both with and without sequencing quality binning (8 bins), obtained using Scramble v1.14.11 (Bonfield, 2014), and also BAM compression (implemented in samtools v1.9; (Li *et al.*, 2009)) and pigz (Adler, 2014). For the reference-based compression methods (i.e. CRAM and genozip --reference) we used a slightly modified version of human reference GRCh37 that was created using the steps described by Luca Santuari (https://github.com/GooglingTheCancerGenome/sv-callers/wiki/Building-the-b37-human-decoy-reference-genome).

FASTQ

FASTQ is a widely used text format that stores sequence data and the corresponding qualities for each nucleotide. We used Bazam (Sadedin and Oshlack, 2019) to generate paired-end FASTQ files from the same BAM file that was used for the VCF benchmark. We compared Genozip against several widely used file compression methods – i.e. gzip, pigz, unaligned BAM, and unaligned CRAM – as well as alignment-based methods that take advantage of sequence similarity to reduce data redundancy (providing similar comparisons to the tools tested in the SAM benchmarks).

The tests were conducted on a Linux machine with 56 cores.

In Table 2 and Figure 3 we describe the compression ratio achieved by Genozip vs other tools, as well as wall time observed.

Table S8: Benchmark files. Uncompressed files used for benchmarking compression of raw (i.e. uncompressed) files against other common tools.

File type	File size	Source
VCF	128 MB	ftp://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/NA12878_HG001/latest/GRCh37/
SAM	147 GB	ftp://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/NA12878/NIST_NA12878_HG001_HiSeq_300x/
FASTQ	2 x 112 GB	Derived from SAM file using Bazam (Sadedin and Oshlack, 2019)

Table S9: Raw-file benchmark results. Results of compression of uncompressed genomic files with genozip and other commonly used tools for each file format.

Tool		Compression	Ratio	Compress time	Decompress t.
	VCF	1,807,059,769			
	.vcf				
pigz	.vcf.gz	113,699,782	15.9	1.9 sec	3.1 sec
bcftools	.bcf	153,988,419	11.7	23.82 sec	21.02 sec
bzip2	.vcf.bz2	71,358,351	25.3	260.05 sec	43.37 sec
genozip	.vcf.genozip	53,819,306	33.6	7.1 sec	6.53 sec
	SAM	510,942,582,641			
pigz	.sam.gz	148,212,447,723	3.4	00:12:40.3	00:34:17.4
samtools	.bam	157,455,536,282	3.2	00:23:16.7	00:29:48.5
scramble -9	.cram	109,288,249,883	4.7	00:27:58.4	00:17:34.4
genozip -e	.sam.genozip	88,329,235,177	5.8	00:33:41.1	00:27:55.3
Optimized cram: scramble -9B	.cram (-B)	85,327,476,246	6.0	00:48:56.1	00:19:10.4
Optimized genozip -9	.sam.genozip (-9)	67,589,616,986	7.6	00:30:51.1	00:20:38.0
	FASTQ	238,958,297,328			
pigz	.fq.gz	57,228,032,622	4.2	00:14:34.5	00:34:17.4
bwa mem | samtools sort | scramble -9	.cram	44,248,758,235	5.4	03:42:54.0	00:48:24.7
genozip -e	.fq.genozip	35,098,350,704	6.8	00:16:40.1	00:08:31.7
genozip -9e	.fq..genozip (-9)	12,837,612,728	18.6	00:08:52.3	00:05:26.4

Compressing against already-compressed files

Table S10: Genozip on already compressed files - Files used

File type	File size	Source
.fastq.gz	3.6 GB (R1+R2)	ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/Garvan_NA12878_HG001_HiSeq_Exome/NIST7035_TAAGGCGA_L001_R1_001.fastq.gz ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/Garvan_NA12878_HG001_HiSeq_Exome/NIST7035_TAAGGCGA_L001_R2_001.fastq.gz
.bam	147 GB	ftp://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/NA12878/NIST_NA12878_HG001_HiSeq_300x/RMNISTHS_30xdownsample.bam
.cram (lossless)	102 GB	Generated from the BAM file with: `scramble -9`
.cram (binned)	79.5 GB	Generated from the BAM file with: `scramble -9 -B`
.vcf.gz	128 MB	ftp://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/NA12878_HG001/latest/GRCh37/HG001_GRCh37_GIAB_highconf_CG-IllFB-IllGATKHC-Ion-10X-SOLID_CHROM1-X_v.3.3.2_highconf_PGandRTGphasetransfer.vcf.gz

Table S11: Genozip on already compressed files. Results of compression with Genozip of already-compressed files in formats in common use in research and medical settings. These results are also reflected in Figure 2 in the main text.

Source file		Genozip command `--optimise` added for the Optimised test	Genozip lossless		Genozip optimised	
File	File size		Size	Factor	Size	Factor
.fastq.gz	3.60 GB	`genozip --pair $file-R1 $file-R2 -e $ref-file`	1.24 GB	2.9 X	0.63 GB	5.7 X
.bam	147 GB	`genozip $file -e $ref-file`	82.3 GB	1.8 X	62.9 GB	2.3 X
.cram (lossless)	101 GB	`genozip $file -e $ref-file`	82.9 GB	1.2 X	63.6 GB	1.6 X
.cram (binned)	79.5 GB	`genozip $file -e $ref-file`	63.6 GB	1.2 X	63.6 GB	1.2 X
.vcf.gz	128 MB	`genozip $file -e $ref-file`	51 MB	2.5 X	50 MB	2.6 X

Compressing BAM vs compressing CRAM

In this test, we compared several aspects of Genozip's performance - compression ratio, time, memory and CPU usage, of compressing a CRAM file vs compressing the same data in BAM format. The tests were run on the same machine as the previous tests - one with 56 cores and over 700GB of RAM.

The CRAM file used was a 14GB file downloaded from ftp://ftp.sra.ebi.ac.uk/vol1/run/ERR324/ERR3241754/HG00731.final.cram and the BAM file used was a 37GB GB file generated from this CRAM file with `samtools view`. The results are in Table S12.

Genozip compresses CRAM files by using `samtools view` to first convert CRAM to SAM. In the decompression step, we piped `genounzip --stdout` into `samtools view -OCRAM` to recreate the CRAM file. The wallclock time in Table S12 represents the combined operation Genozip and `samtools`, while the CPU time and memory reflect only Genozip's, and not samtools', resource consumption.

In contrast, Genozip compresses and decompresses BAM files natively, without relying on `samtools` or `htslib`. Consequently, Genozip is free to scale to a much larger number of CPUs and complete the processing faster. The higher memory consumption in the BAM case (Table S12) is a reflection of Genozip's ability to scale to a larger number of CPU cores, and hence threads, in this case. The higher CPU time is mostly due to Genozip also decompressing the BAM BGZF compression (in `genozip`) and recreating BAM in compressed BGZF format (in `genounzip`) which it does not do in the case of CRAM, because the plain SAM data is piped in from or piped out to `samtools`.

Table S12: Compressing CRAM vs compressing BAM. Results showing Genozip's performance when compressing CRAM and BAM files containing identical data. With BAM, Genozip can scale to a larger number of CPUs.

	CRAM - compress	**BAM - compress**	**CRAM - decompress**	**BAM - decompress**
Orig file size	14482707829	38828072041		
Compressed size	13190514012	13530076092		
Ratio	1.1 X	2.9 X		
Wall clock time	14m 9s	7m 57s	13m 40s	5m 52s
CPU time	19017 sec	23211	8076 sec	18188
Max memory	14.5 GB	20.8 GB	10.8 GB	12.4 GB
CPUs utilized	22.4	48.7	9.8	51.6

Chapter 3

Genozip Dual-Coordinate VCF format enables efficient genomic analyses and alleviates liftover limitations

Divon Lan[1,*], Gludhug Purnomo[1,2], Ray Tobler[1,2,3, †], Yassine Souilmi[1,4, †], Bastien Llamas[1,2,4,5,†,*]

[1] Australian Centre for Ancient DNA, School of Biological Sciences, The Environment Institute, Faculty of Sciences, The University of Adelaide, Adelaide SA 5005, Australia
[2] Centre of Excellence for Australian Biodiversity and Heritage (CABAH), School of Biological Sciences, University of Adelaide, Adelaide, SA 5005, Australia
[3] Evolution of Cultural Diversity Initiative, Australian National University, College of Asia and the Pacific, Canberra, ACT 0200, Australia
[4] National Centre for Indigenous Genomics, Australian National University, Canberra, ACT 0200, Australia
[5] Indigenous Genomics Research Group, Telethon Kids Institute, Adelaide, SA 5000, Australia

[†] Equal contribution
* Correspondence: DL (divon@genozip.com) and BL (bastien.llamas@adelaide.edu.au)

Statement of Authorship

Title of Paper	Genozip Dual-Coordinate VCF format enables efficient genomic analyses and alleviates liftover limitations
Publication Status	☐ Published ☐ Accepted for Publication ☑ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	

Principal Author

Name of Principal Author (Candidate)	Divon Mordechai Lan		
Contribution to the Paper	Wrote the software and the draft manuscript		
Overall percentage (%)	90%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature	[illegible]	Date	20/04/2022

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);
ii. permission is granted for the candidate in include the publication in the thesis; and
iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Gludhug Purnono		
Contribution to the Paper	Provided feedback that helped improve the software		
Signature		Date	21/04/2022

Name of Co-Author	Yassine Souilmi		
Contribution to the Paper	Reviewed and edited the manuscript and provided guidance		
Signature		Date	20/04/2022

Name of Co-Author	Ray Tobler		
Contribution to the Paper	Reviewed and edited the manuscript and provided guidance		
Signature		Date	20/04/2022

Name of Co-Author	Bastien Llamas		
Contribution to the Paper	Reviewed and edited the manuscript and provided guidance		
Signature		Date	20/04/2022

Abstract

We introduce Dual Coordinate VCF (DVCF), a file format that records genomic variants against two different reference genomes simultaneously and is fully compliant with the current VCF specification. As implemented in the Genozip platform, DVCF enables bioinformatics pipelines to seamlessly operate across two coordinate systems by leveraging the system most advantageous to each pipeline step, simplifying bioinformatics workflows and reducing file generation and associated data storage burden. Moreover, our benchmarking of Genozip DVCF shows that it produces more complete, less erroneous, and less biased translations across coordinate systems than two widely used alternative tools (i.e., LiftoverVcf and CrossMap).

Availability and Implementation: Genozip is free for academic use. Documentation is available on https://genozip.com/dvcf.html. Genozip user manual is available on https://genozip.com/manual.html. The source code is available on https://genozip.com/source.html. The scripts for reproducing the benchmarks are available on https://github.com/divonlan/genozip-dvcf-results.

Main

Genomic sequencing and assembly technologies continue to evolve at a rapid pace, enabling the creation of new and more accurate reference genomes for various species [1]. While improved human reference genomes are welcomed by researchers and clinicians, updated assemblies inevitably result in altered coordinates that can hinder their adoption as it is common that legacy datasets and bioinformatics software required for some analyses often still use the older coordinate system. Accordingly, for species such as humans where multiple reference genomes (or versions thereof) are available, analytical pipelines often need to alternate between two coordinate systems to accommodate these limitations.

To facilitate workflows involving data mapped against different reference genomes, software such as CrossMap [2] and GATK LiftoverVcf [3] translate genomic coordinates across the assemblies using a chain file [4]. Variants, typically encoded in a VCF (Variant Call Format) [5] file, are then lifted over from one coordinate system to the other, resulting in file duplication (i.e., one for each coordinate system). The liftover step is typically lossy, with variants discarded due their coordinates lacking alignment in the chain file as well as limitations of the liftover software used [1, 6]. Moreover, errors may be introduced due to incorrect chain file mapping of variants as well as incorrect annotation conversion, along with the introduction of potential biases due to the concentration of discarded variants in certain genomic regions.

Here we introduce an implementation of the *Dual Coordinate VCF* (DVCF) [7] format in the Genozip platform [8, 9], an extensible compression software. DVCF is an extension to the standard VCF format compliant with the VCF 4.3 specification, that includes variants with coordinates pertaining to two different genome assemblies simultaneously (Fig. S1). Using DVCF files, researchers can alternate between coordinate systems according to their needs – without creating duplicate VCF files – thereby reducing workflow complexity and alleviating demands on time, computational resources, and disk storage burden (the latter being further improved by Genozip's efficient data compression algorithms; Tables S16-17). Importantly, the DVCF file format is independent of its implementation in Genozip, allowing its implementation in any relevant bioinformatics software, thereby maintaining interoperability between tools.

We refer to the process of converting a VCF file to the DVCF format as *lifting.* In order to lift a VCF file to DVCF, Genozip requires three inputs: two reference files – one defining the *Primary* coordinates as used in the input VCF, and the other defining the *Luft* coordinates to be lifted over ("*luft*" is a neologism representing an alternative past-participle of "lift") – and a chain file defining alignments between the two reference assemblies. Once a DVCF file is generated, users can render it in either *Primary* or *Luft* coordinates by using Genozip's `genocat` command (see Figs. 1A, S2). *Cross-rendering* consists of losslessly re-arranging the information in the DVCF file to change the coordinate system in which the variants are represented. This allows users to seamlessly alternate between coordinate systems according to the particulars of their bioinformatic workflow.

While a DVCF has two possible renditions (*Primary* and *Luft*), the DVCF file format is carefully designed so that the information contained in each rendition is identical, thereby guaranteeing that the *cross-rendering* process is strictly lossless and invertible. Variants or annotations that have representation in only one of the two coordinate systems due to the lack of a chain file alignment or limitations of the lifting algorithm are nevertheless still represented in both DVCF *renditions.* The missing variants and annotations are stored in VCF header lines and certain INFO annotations, respectively, as defined in the DVCF specification (sections 5,6,7 in the specification [7]). Since the DVCF file format complies with the VCF standard, any tool that works with VCF files will also work with DVCF files. In addition to rendering the complete VCF data, Genozip's `genocat` command allows users to render specific subsets of data in either coordinate system, as well as perform a wide variety of downsampling and filtering procedures, by drawing upon Genozip's internal indexing facility (see the `genocat` user manual).

To compare the performance of Genozip DVCF against two widely used liftover tools, i.e., LiftoverVcf and CrossMap, we conducted a series of benchmarks using publicly available human genomic data (Supplementary Information section 2). For each file tested, we used the same chain file with all three tools, but nevertheless, each of the three tools produced a different lifted VCF file. We systematically investigated the differences in the lifted files produced by the three tools, characterising and enumerating errors and biases that result from underlying deficiencies in the algorithms of each liftover tool. In other words, we did not evaluate the correctness of the chain files, but rather the correctness of the lifting algorithms for any given chain file.

Genozip outperformed both incumbent tools when lifting a set ~19,000 indels, or ~4.1 million SNPs from an older version of the human reference (i.e., GRCh37) to the most recent version (i.e., GRCh38), reducing the proportion of incorrect calls by nine-fold for indels (0.2% vs 2.1–2.3%) and 463-fold for SNPs (0.002% vs 1.1%) (Fig. 1B, Tables S1-11). Notably, CrossMap also dropped all instances of SNPs where the reference and alternate alleles had been switched between the two reference versions (i.e., REF⇄ALT switches, which accounted for 0.7% of all 4.1M SNPs), which may introduce biases into downstream analyses that leverage SNP diversity patterns due to the highly non-uniform genomic distribution of the REF⇄ALT switches (Figs. 1C, S3-4). We also applied the three tools to ~970k clinically relevant variants obtained from the ClinVar website [10] and identified multiple cases of data corruption introduced by CrossMap and LiftoverVcf, which included known pathogenic variants, that are entirely absent from Genozip DVCF (Fig. 1D; Tables S13-15).

In conclusion, the implementation of DVCF within the Genozip software platform provides researchers with a user-friendly and flexible tool that facilitates the construction of bioinformatic pipelines capable of working across dual coordinate systems. This is essential whenever researchers wish to exploit the advantages of working with sequence data aligned to the latest version of the reference genome, while still being able to draw upon abundant legacy data or tools that use an older reference genome version.

Genozip DVCF also correctly lifts more variants and eliminates key errors identified in our benchmarks of two widely used liftover tools, CrossMap and LiftoverVcf. Failure of these latter tools to correctly liftover variants in regions of the genome causally associated with phenotypes (Fig. 1D) could negatively impact genetic analyses that rely on regional genomic signals—such as genomic scans for disease-associated and/or selected variants. Moreover, liftover errors have documented impacts on variant effect interpretation [6], which could result in important clinically significant variants being overlooked or leading to misdiagnoses [1].

Overall, DVCF represents a new and fundamentally different approach for working concurrently within coordinate systems from two different genome assemblies—a reality that many genomics researchers will likely face as improved sequencing technologies lead to increasingly complete reference genomes [1]—greatly simplifying bioinformatic workflows without compromising the robustness of downstream analytical results. Importantly, Genozip's extensible framework means that further improving DVCF functionality (e.g., by including new algorithms to handle liftover errors, or supporting an arbitrary number of reference genomes, thereby enabling recording of homologous genes in different species) will be an active area of future research.

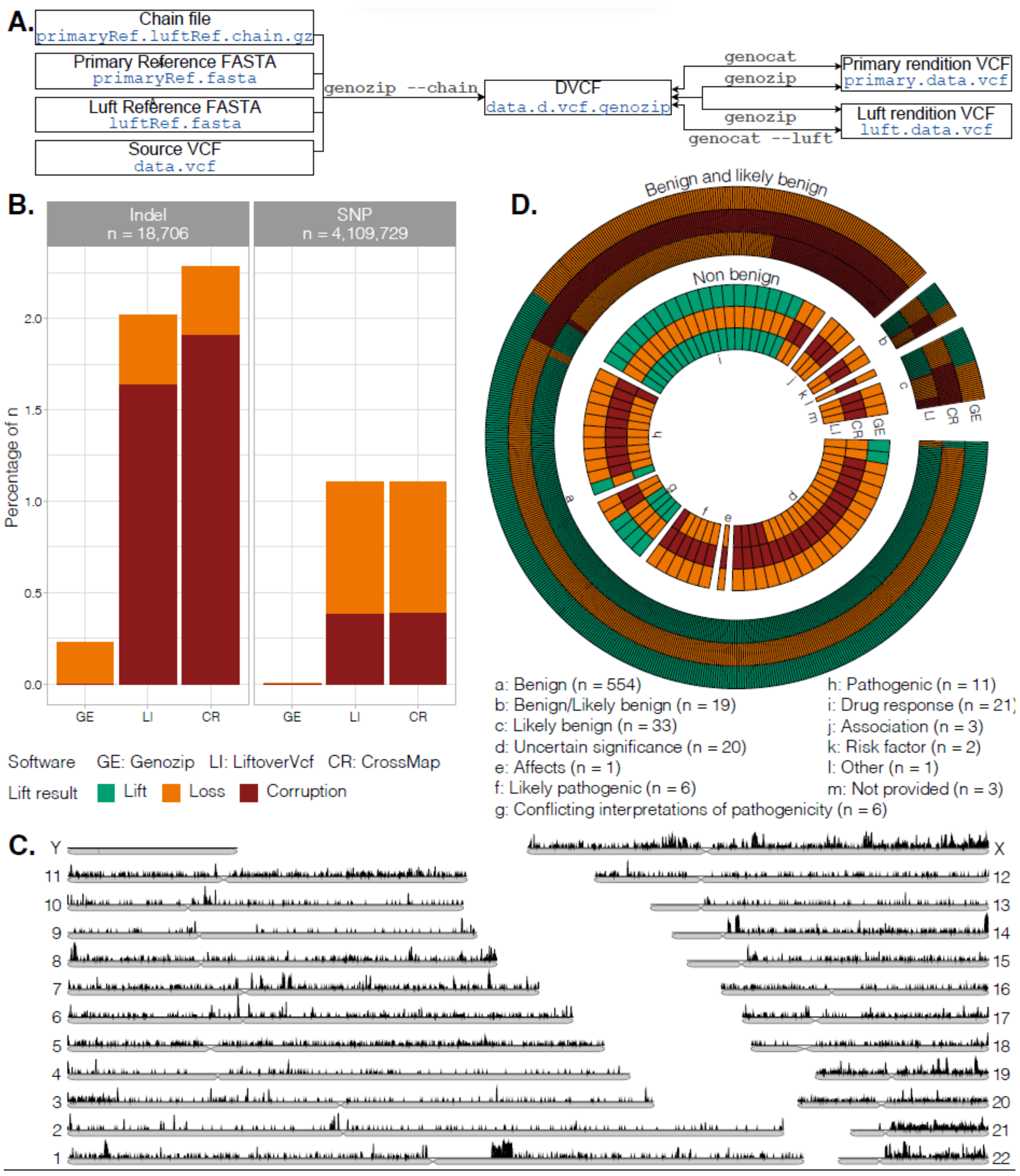


Figure 1. DVCF command line usage and performance vs. CrossMap and LiftoverVcf. **A.** Generation of primary (*Primary*) and alternate (*Luft*) renditions and their integration into a bioinformatics pipeline. Commands are shown along the connecting arrows and file names are indicated in blue text. File suffixes are automatically generated by genozip. **B.** When lifting ~4.1M SNPs and ~19k indels from GRCh37 to GRCh38 (right and left subpanels, respectively), Genozip DVCF results in substantially fewer lost variants (orange bar portions) and eradicates all forms of data corruption (red bar portions) (see Tables S1 & S9). **C.** Genomic distribution of the ~30k SNPs (from a set of ~4.1M SNPs) where the reference and alternative allele are switched (i.e. REF⇄ALT allele switches) between human reference GRCh37 and GRCh38. While Genozip DVCF correctly lifted all ~30k SNPs, CrossMap

drops these variants (see Table S9), which may lead to biases in downstream analyses due to the genomic clustering of REF⇄ALT allele switches. **D.** Similar results are observed when lifting a set ~970k variants with clinical annotations from GRCh37 to GRCh38, with CrossMap and LiftoverVcf producing many more dropped variants (orange blocks) than Genozip DVCF and also generating multiple corrupted variants (red blocks) – including pathogenic and likely pathogenic cases (see associated key) – that are entirely absent from Genozip DVCF (see tables S13-15).

References

1. Aganezov S, Yan SM, Soto DC, Kirsche M, Zarate S, Avdeyev P, et al. A complete reference genome improves analysis of human genetic variation. bioRxiv. 2021;:2021.07.12.452063.
2. Zhao H, Sun Z, Wang J, Huang H, Kocher J-P, Wang L. CrossMap: a versatile tool for coordinate conversion between genome assemblies. Bioinformatics. 2014;30:1006–7.
3. Broad Institute. Picard tools. {Broad Institute, GitHub repository}.
4. Chain Format. https://genome.ucsc.edu/goldenPath/help/chain.html. Accessed 23 Feb 2022.
5. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, et al. The variant call format and VCFtools. Bioinformatics. 2011;27:2156–8.
6. Ormond C, Ryan NM, Corvin A, Heron EA. Converting single nucleotide variants between genome builds: from cautionary tale to solution. Brief Bioinform. 2021. https://doi.org/10.1093/bib/bbab069.
7. Lan D. The variant call format - Dual Coordinate extension (DVCF) specification. figshare; 2021.
8. Lan D, Tobler R, Souilmi Y, Llamas B. Genozip - A Universal Extensible Genomic Data Compressor. Bioinformatics. 2021. https://doi.org/10.1093/bioinformatics/btab102.
9. Lan D, Tobler R, Souilmi Y, Llamas B. genozip: a fast and efficient compression tool for VCF files. Bioinformatics. 2020;36:4091–2.
10. Landrum MJ, Lee JM, Benson M, Brown G, Chao C, Chitipiralla S, et al. ClinVar: public archive of interpretations of clinically relevant variants. Nucleic Acids Res. 2015;44:D862–8.

Acknowledgements

D.L. was supported by a scholarship from the University of Adelaide. Y.S. is supported by the Australian Research Council (DP190103705). R.T. is an ARC DECRA fellow (DE190101069). B.L. was an ARC Future Fellow (FT170100448).

Conflict of Interest: D.L. intends to receive royalties from commercial users of genozip.

Contributions

DL wrote the software and the first draft of the manuscript. BL, RT and YS reviewed and edited the manuscript and provided guidance. GP provided feedback that helped improve the software.

Corresponding author

Correspondence to Divon Lan (divon@genozip.com) and Bastien Llamas (bastien.llamas@adelaide.edu.au).

Supplementary Information

Supplementary Information
Supplementary text
Supplementary Figs. S1-S4
Supplementary Tables S1-S20

Genozip Dual-Coordinate VCF format enables efficient genomic analyses and alleviates liftover limitations

Supplementary Information

Divon Lan[1,*], Gludhug Purnomo[1,2], Ray Tobler[1,2,3,†], Yassine Souilmi[1,4, †], Bastien Llamas[1,2,4,5,†,*]

[1] Australian Centre for Ancient DNA, School of Biological Sciences, The Environment Institute, Faculty of Sciences, The University of Adelaide, Adelaide SA 5005, Australia

[2] Centre of Excellence for Australian Biodiversity and Heritage (CABAH), School of Biological Sciences, University of Adelaide, Adelaide, SA 5005, Australia

[3] Evolution of Cultural Diversity Initiative, Australian National University, College of Asia and the Pacific, Canberra, ACT 0200, Australia

[4] National Centre for Indigenous Genomics, Australian National University, Canberra, ACT 0200, Australia

[5] Telethon Kids Institute, Adelaide, SA 5000, Australia

[†] Equal contribution

* Corresponding authors: DL (divon@genozip.com) and BL (bastien.llamas@adelaide.edu.au)

SI.1. DVCF Implementation in Genozip

1.1 The DVCF format

DVCF files are a type of VCF file, compliant with the VCF v4.3 specification, which contain information about variants represented in two different coordinate systems. Like any VCF file, it consists of meta-information lines prefixed with a double hash (i.e. ##), a header line prefixed with #CHROM, and data lines that each contain information about a single variant (Figure S1). Importantly, the DVCF specification[6] only defines the DVCF file format and does not prescribe liftover algorithms, making it independent of any specific software implementation (including Genozip) and we expect other bioinformatics software packages to implement DVCF as well, thereby maintaining interoperability between tools. For example, existing liftover tools could add a command line option that enables the generation of DVCF formatted outputs.

We refer to the process of converting a VCF file to the DVCF format as *lifting. Lifting* requires information external to the VCF file itself, in a format defined by the implementation. The DVCF implementation in Genozip, for example, requires two reference files—i.e. the reference file that underlies the coordinate system used in the VCF, and the alternate reference file that contains the coordinates to be lifted over—and a chain file describing the mapping of variants between these two systems, but future implementations could work differently. The coordinate system of the input VCF file is referred to as the *Primary* coordinates, and the *lifting* process updates this VCF by incorporating the required information from the alternate coordinate system, which we refer to as the *Luft* coordinate system (*Luft* being a term we introduce as an alternative past-participle of *Lift*).

Thereafter, a DVCF can be *rendered* in either *Primary* coordinates or *Luft* coordinates, and can be *cross-rendered* from *Primary* to *Luft* coordinates and vice versa. We refer to the DVCF-format VCF files rendered in the *Primary* coordinates as the *Primary rendition* and to the corresponding VCF file in the *Luft* coordinates as the *Luft rendition.* The coordinate system in which a VCF file is rendered (i.e. the coordinates represented in the CHROM and

POS fields) is referred to as the *foreground* coordinate system (which may be *Primary* or *Luft*), with the non-rendered coordinate system being the *background* coordinate system.

Crucially, the information present in both *foreground* and the *background* coordinate systems is present in both renditions: in particular, the CHROM and POS fields are encoded in *foreground* coordinates and the allelic states represented in the REF and ALT fields are defined relative to the *foreground* reference genome. Simultaneously, the *background* coordinate data are encoded in the INFO/LUFT or INFO/PRIM annotation in the *Primary* or *Luft* rendition respectively (Figure S1, notes 5 and 6) for each variant and also in the meta information section (Figure S1, notes 1–5).

Annotations for some variants may differ between the *Primary* and *Luft* renditions – for example, annotations that are sensitive to a change in the reference allele (e.g. INFO/AF), or to reference file strand reversal (e.g. INFO/BaseCounts) or coordinate changes (e.g. INFO/END), will only be correct with respect to the *foreground* coordinate system. Cross-rendering of variants with rendition-specific annotations is handled by *Rendering Algorithms* (or *RendAlgs*), which are a set of algorithms that convert specific annotations between renditions to ensure consistency with the foreground coordinates. Several standard RendAlgs are defined in section 6.3 of the DVCF specification and implementation developers are encouraged, but not mandated, to use the standard RendAlgs wherever possible, but may also develop proprietary RendAlgs.

There are cases where the tag name itself, rather than the values contained in the annotation, differs between renditions—for example, the FORMAT/ADR and FORMAT/ADF tags might switch names (ADF becomes ADR and vice versa) in variants that have a reference genome strand reversal. Another example of using tag renaming would be in case dropping a particular annotation is desired. In this case, the tag name can be prefixed with DROP_. The DVCF specification defines three *Tag Renaming* attributes (see section 6.4 of the DVCF specification).

A key benefit of the DVCF file format design is that *cross-rendering* consists of simply rearranging information that is already present in the DVCF file and transforming affected annotations, and therefore this process does not require external information (such as the reference sequence or the chain files), making it computationally fast and efficient. Moreover, since DVCF files are VCF files, they can be processed in the same manner as traditional VCF files using appropriate bioinformatics tools, but provide users with the additional freedom to choose which coordinate system to use when executing specific pipeline steps without having to create duplicate VCF files, thereby reducing analytical complexity and saving considerable disk space (SI section 4).

A likely outcome of the lifting process is that some variants may only be represented in one of the two coordinate systems due to the lack of a specific chain file alignment or liftover software limitations. To maintain information identicality across renditions, such *single-coordinate* variants are represented in both renditions, by recording their *background* coordinates in the VCF meta-information (prefixed with either `##primary_only=` or `##luft_only=` in accordance with the coordinate system that the variant is represented; see Figure S1, note 1). We refer to such variants as *Primary-only* or *Luft-only* variants depending on which reference version the variant is represented. In addition, meta information lines describing the contigs and reference file of the *background* coordinate system may also be added (DVCF specification section 5.3 and 5.6 ; Figure S1, note 3).

In conclusion, while a DVCF has two *renditions* (i.e. *Primary* and *Luft*), the DVCF file format is carefully designed so that the information contained in each *rendition* is identical, thereby guaranteeing that the *cross-rendering* between the *renditions* is a strictly lossless and invertible process.

```
[1]##primary_only=1        143163348       LN=170280       G       A       1066.01
.       AC=1;AF=0.500;AN=2;Lrej=NoMappingInChainFile   GT:AD:DP:GQ:PL:FL
0/1:152,74:1066,0,2824
[2]##contig=<ID=chr1,length=248956422>
[2]##reference=file://GRCh38_full_analysis_set_plus_decoy_hla.ref.genozip
[3]##primary_reference=file://hs37d5.ref.genozip
[3]##primary_contig=<ID=1,length=249250621>
[4]##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref
and alt alleles in the order listed",RendAlg=R>
[4]##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype",RendAlg=GT>
[4]##FORMAT=<ID=PL,Number=G,Type=Integer,Description="Normalized, Phred-scaled
likelihoods for genotypes as defined in the VCF specification",RendAlg=G>
[4]##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes, for
each ALT allele, in the same order as listed",RendAlg=A_AN>
[4]##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency, for each ALT
allele, in the same order as listed",RendAlg=A_1>
[4]##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in
called genotypes",RendAlg=NONE>
[5]##INFO=<ID=LUFT,Number=4,Type=String,Description="Info for rendering variant in
LUFT coords. See
https://genozip.com/dvcf.html",Source="genozip",Version="12.0.8",RendAlg=NONE>
[5]##INFO=<ID=PRIM,Number=4,Type=String,Description="Info for rendering variant in
PRIMARY coords",Source="genozip",Version="12.0.-1",RendAlg=NONE>
[5]##INFO=<ID=Lrej,Number=1,Type=String,Description="Reason variant was rejected
for LUFT coords",Source="genozip",Version="12.0.-1",RendAlg=NONE>
[5]##INFO=<ID=Prej,Number=1,Type=String,Description="Reason variant was rejected
for PRIMARY coords",Source="genozip",Version="12.0.-1",RendAlg=NONE>
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  SS6004478
[6]chr1    597733  LN=298  A       G       263.03  .
AC=2;AF=1.00;AN=2;PRIM=1,533113,A,-   GT:AD:PL       1/1:4,20:263,24,0
```

Figure S1. A small subset of the Luft rendition of our SNP test file (see section 4), containing two variants. Notes: 1a Primary-only variant represented in a ##primary_only line. 2foreground (Luft in this case) coordinate data 3background (Primary) meta-information lines are preserved with the prefix primary_. 4INFO and FORMAT meta-information lines contain the RendAlg attribute 5Four DVCF INFO tags defined. 6PRIM contains the data needed to cross-render to Primary.

1.2 Genozip – a brief overview

Genozip is a software platform that stores genomic files in a compressed format. For VCF files, compression is typically 2–8 times better than other standard compression tools such as gzip and BCF (Lan *et al.*, 2020). In Genozip's implementation of DVCF files, data are stored in the Genozip format. Users may render the data in either coordinate system and have the option of piping the data through an analytical tool or pipeline before returning the data to Genozip format (see Figure 1A). In the following section, we outline the basic commands involved in the *lifting* process and provide a brief description of the algorithms involved in *cross-rendering* between coordinate systems.

1.3 The *lift* process

Genozip can lift one or more VCF files into a DVCF using a suitable chain file available from Ensembl (https://ftp.ensembl.org/pub/assembly_mapping/) or the UCSC Genome Browser (https://hgdownload.soe.ucsc.edu/downloads.html) by invoking the following command:

```
genozip --chain mychainfile.chain.genozip myvariants.vcf
```

This will generate a DVCF file in Genozip format: myvariants.d.vcf.genozip.

Genozip accepts chain files in the UCSC chain format (genome.ucsc.edu/goldenPath/help/chain.html), which must first be compressed with `genozip` using Primary and Luft reference files:

```
genozip --reference primary.ref.genozip --reference
luft.ref.genozip mychainfile.chain
```

Each reference file, in turn, is produced from the relevant reference genome FASTA file:

```
genozip --make-reference primary.fa.gz
```

During the lift process (i.e. executing `genozip --chain`), Genozip determines whether it is possible to represent each variant in both coordinate systems, with all variants failing this process (i.e. rejected variants) being retained as Primary-only variants. A variant may be rejected for three reasons: (1) it lacks an alignment in the chain file, or (2) the lifted variant has more than two alleles (either because it had more than two alleles in the input VCF file, or the Luft REF allele is neither the Primary REF nor ALT allele), or (3) if any of the INFO or FORMAT annotations cannot be cross-rendered due to the “Rejected If” condition of their RendAlg (see next section). Known cases in which Genozip cannot lift variants are listed in genozip.com/dvcf-limitations.html.

1.4 Rendering in *Primary* or *Luft* coordinates

When in the Genozip format, DVCF files may be rendered in Primary or Luft coordinates by using either `genocat data.vcf.genozip` or `genocat --luft data.vcf.genozip`, respectively (Figure S2). The resulting VCF file is sorted according to its foreground (i.e., rendered) coordinates. Running `genozip` on the rendered file in either coordinate system produces a compressed DVCF file in Genozip format containing identical information.

In addition to variants becoming single coordinate during the initial lift process, they can also become single coordinate when compressing a DVCF rendition with the `genozip` command if (1) new variants were added to the VCF following the initial lifting step and these variants lack the INFO/PRIM or INFO/LUFT fields (i.e. they are single-coordinate), or (2) if an annotation was added or modified in a way that satisfies its RendAlg's "Rejected If" condition (see next paragraph).

Since both renditions are DVCF files, each rendition contains all the information needed for rendering in either coordinate system. Cross-rendering involves the translation of annotations between the Primary and Luft coordinate systems. This process often implicates making rendition-specific changes to one or more annotations for some variants, with each annotation being handled by a Rendering Algorithm (or RendAlg). RendAlgs are stand-alone algorithms that are not tied a priori to specific INFO and FORMAT tags, rather, they are assigned to tags using a new RendAlg attribute added to each ##INFO and ##FORMAT meta-information line. Thereafter, all INFO and FORMAT annotations (in the VCF data lines) of a specific tag will be treated with the `RendAlg` assigned to that tag. Importantly, Genozip adds the `RendAlg` attribute to each tag's meta-information line based on its ID and/or Number attributes, but only if the `RendAlg` attribute is not already present. This allows users to assign RendAlgs differently than Genozip's built-in assignments.

Genozip implements eleven RendAlgs, listed in genozip.com/dvcf-rendering.html, which are an implementation of the DVCF standard set of RendAlgs defined in the specification6section 6.3; see also genozip.com/dvcf-rendering.html). Each RendAlg is comprised of three elements: (1) a trigger; which is the class of event handled by the RendAlg, (2) an action which is the effect the RendAlg has on the annotation, and (3) "Rejected If" conditions, which describe circumstances in which the designated action cannot be applied despite the trigger activating, in which case the variant is rejected (i.e., is represented as a single-coordinate variant in the DVCF). To illustrate this mechanism, consider the A_1 RendAlg which is useful for annotations in fields reporting allele frequency information. The A_1 RendAlg trigger is a "REF⇄ALT switch" (i.e., a change of REF allele across renditions), its action is to recalculate the value of the annotation as 1 - value, and the variant is "Rejected If" the value of the annotation is outside of the range [0,1] or if the variant has more than two alleles. By default, Genozip assigns the A_1 RendAlg to the `FORMAT/AF`, `INFO/AF`, `INFO/MLEAF`, `INFO/LDAF` fields and any additional INFO tag that begins with `AF_` or ends with `_AF` (except `MAX_AF`). Another example is the RendAlg named END that Genozip assigns by default to the `INFO/END` field. Its trigger event is "Always" (i.e., it triggers on all `INFO/END` annotations in the data), its action is to modify the annotation to which it is assigned to maintain the same distance from POS in both renditions, whereby the variant is "Rejected If" the position indicated by the annotation and POS are not both on the same chain file alignment. A final example is the XREV RendAlg whose trigger event is "strand reversal" (as indicated by the chain file alignment) and its action is to reverse the elements of an array. Genozip assigns the XREV RendAlg to the `INFO/BaseCounts` field.

```
> genocat SS.d.vcf.genozip -H -s 1 -g LN=632  # PRIMARY RENDITION
1       770568  LN=632  A       G       809.01  .
AC=2;AF=1.00;AN=2;BaseCounts=0,0,31,1;DB;DP=32;Dels=0.00;FS=0.000;GC=47.1
3;HaplotypeScore=0.0000;MLEAC=2;MLEAF=1.00;MQ=32.04;MQ0=4;QD=25.28;LUFT=c
hr1,835188,G,- GT:AD:DP:GQ:PL:FL       1/1:0,31:31:63:809,63,0:N



> genocat SS.d.vcf.genozip -H -s 1 -g LN=632 --luft # LUFT RENDITION
chr1    835188  LN=632  G       A       809.01  .
AC=0;AF=0.00;AN=2;BaseCounts=0,0,31,1;DB;DP=32;Dels=0.00;FS=0.000;GC=47.1
3;HaplotypeScore=0.0000;MLEAC=0;MLEAF=0.00;MQ=32.04;MQ0=4;QD=25.28;PRIM=1
,770568,A,-    GT:AD:DP:GQ:PL:FL       0/0:31,0:31:63:0,63,809:N
```

Figure S2. An example of rendering and cross rendering with one variant. The differences between the Primary rendition (top command line and output) and Luft rendition (bottom command line and output) are highlighted in bold font. Notice that the LUFT and PRIM subfields of the INFO field display the information that is used by Genozip to cross-render this variant.

1.5 Disk space considerations

In addition to needing only a single DVCF file to represent variants in two coordinate systems, rather than two separate VCF files, Genozip also stores VCF files using highly efficient compression (Lan et al. 2020). Together, this amounts to significant saving of disk space.

To demonstrate this, we compare the size of our SNP and Indel test files (SI section 1):

Table S16: SNP file

	Uncompressed	**.gz compressed**	**Genozip DVCF**
GRCh37[1]	999 MB	199 MB	76 MB
GRCh38[2]	1094 MB	175 MB	
Total	**2093 MB**	**374 MB**	**76 MB**
Compression ratio	**-**	**5.6X**	**27.5X**

[1]SS6004478.annotated.nh2.variants.vcf [2]snp.38.gatk.vcf

Table S17: Indel file

	Uncompressed	**.gz compressed**	**Genozip DVCF**
GRCh37[1]	3961 KB	848 KB	443 KB
GRCh38[2]	4470 KB	777 KB	
Total	**8431 KB**	**1625 KB**	**443 KB**
Compression ratio		**5.2X**	**19X**

[1]indel.37.vcf [2]indel.38.gatk.vcf

SI.2. Benchmark

2.1 Categorizing variants by lift quality

We used chromosome 22 data from the 1000 Genome project phase 1 for benchmarking indels (see Section 1.2) and a sample from https://sharehost.hms.harvard.edu/genetics/reich_lab/sgdp/vcf_variants/ for benchmarking SNPs (see Section 1.3).

We categorize the 18,857 indel variants in the indel test file and the 4,109,729 variants in the SNP test file, in respect to *lifting*—the operation of converting the VCF file from one coordinate system to another, GRCh37 to GRCh38 in our case.

With respect to each of the three tools, Genozip, LiftoverVcf and CrossMap, we assign one of five categories to each variant. Numbers 1 and 2 are good outcomes, and numbers 3 through 5 are bad outcomes, with increasing order of severity.

1) ***Lifted*** - The lift operation succeeded and the resulting variant is correct.

2) ***Unmapped*** - The chain file has no mapping for the coordinate of this variant, and therefore the variant was correctly rejected from lifting by the tool.

3) ***Annotation Loss*** - The variant was lifted and the resulting variant is correct but incomplete—some of the annotations it originally contained were dropped by the tool.

4) ***Variant Loss*** - The variant was rejected from lifting by the tool, despite having all the information needed to successfully lift it. This happens when variants have some complexities that are beyond the capabilities of the particular tool.

5) ***Data Corruption*** - The variant was lifted, but the resulting variant contains incorrect data.

In some cases, we decided to categorize a variant as *Lifted* even though it had a very *Minor data loss*. This was done in cases where data loss almost certainly will not affect any downstream analysis. These cases are explained where they occur in the sections below.

2.2 Indel benchmark

2.2.1 Data preparation

We developed a set of scripts that executes this benchmark in its entirely - from downloading the data to producing the analysis files. It is available from https://github.com/divonlan/genozip-dvcf-results and the entry point script is:
run-indl-37-38.sh

The key steps executed by this script are these:

Preparing the input files

The indel test file was generated from the chromosome 22 VCF file of the 1000 Genome Project phase 1 obtained from ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/20110521/ALL.chr22.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz and filtered to contain only its indel variants (N=18,706 indels).

The GRCh37 reference was obtained from ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/phase2_reference_assembly_sequence/hs37d5.fa.gz, and prepared for Genozip use with

```
> genozip --make-reference hs37d5.fa.gz
```

The GRCh38 reference was downloaded from ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/GRCh38_reference_genome/GRCh38_full_analysis_set_plus_decoy_hla.fa, and prepared for Genozip use with:

```
> genozip --make-reference GRCh38_full_analysis_set_plus_decoy_hla.fa.gz -o
GRCh38.ref.genozip
```

The chain file mapping GRCh37 genomic features to their corresponding GRCh38 coordinates was obtained from http://ftp.ensembl.org/pub/assembly_mapping/homo_sapiens/GRCh37_to_GRCh38.chain.gz , and prepared with:

```
> genozip --echo GRCh37_to_GRCh38.chain.gz --reference hs37d5.ref.genozip
--reference GRCh38.ref.genozip --match-chrom-to-reference --force --output
GRCh37_to_GRCh38.matched.chain.genozip
```

Note the `--match-chrom-to-reference`: this modifies the contig names in the chain file to match those of the reference files (e.g. “22” vs “chr22”).

For our indels-test, we used a single sample, indels-only VCF file that was generated with the following commands:

```
> genozip
ALL.chr22.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz

> genocat
ALL.chr22.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.genozip
--samples 1 --indels-only -o indel.37.vcf
```

Genozip DVCF liftover

Using the chain file, which itself uses the two reference files, we generated a DVCF file. For the purpose of ease of comparison between corresponding VCF files in the two coordinate systems, we replaced the ID field with a line number (e.g. “LN=452”) using the `--add-line-numbers` command line option, and modified the contig names to match those of the reference file and the chain file with `--match-chrom-to-reference`.

```
> genozip --echo --chain
GRCh37_to_GRCh38.matched.chain.genozip --add-line-numbers
--match-chrom-to-reference indel.37.vcf -o
indel-37-38/indel-37-38.d.vcf.genozip
```

In order to test the other tools with the VCF file that contains the updated ID field and contig names, and also to allow is comparison of Genozip DVCF status to the outcome of the other tools, we generated a GRCh37-only version of the VCF that contains the DVCF lifting status (which we refer to as “oStatus”). The `--single` option converts the dual-coordinate DVCF file to a single-coordinate (GRCh37 in this case) VCF file. `--show-ostatus` adds an `INFO/oSTATUS` field to each variant, describing the Genozip’s liftover status of this variant. The ID field already contains the line number.

```
> genocat indel-37-38.d.vcf.genozip --single -o
indel.37.annotated.vcf --show-ostatus
```

CrossMap liftover

CrossMap.py version v0.5.2 was installed using conda and the test VCF file was lifted over to GRCh38 coordinates using the following command:

```
> CrossMap.py vcf GRCh37_to_GRCh38.matched.chain
indel-37-38/indel.37.annotated.vcf
GRCh38_full_analysis_set_plus_decoy_hla.fa.gz
indel-37-38/indel.38.CrossMap.vcf
```

GATK LiftoverVcf liftover

GATK version 4.1.7 was used, and the indel test file lifted over to GRCh38 coordinates was produced using `LiftoverVcf`:

```
> gatk --java-options '-Xmx16g -XX:ParallelGCThreads=1'
LiftoverVcf --INPUT indel-37-38/indel.37.annotated.vcf
--OUTPUT indel-37-38/indel.38.gatk.vcf --CHAIN
GRCh37_to_GRCh38.matched.chain --REJECT
indel-37-38/indel.38.gatk.rejects.vcf
--RECOVER_SWAPPED_REF_ALT --REFERENCE_SEQUENCE
GRCh38_full_analysis_set_plus_decoy_hla.fa.gz
--TAGS_TO_REVERSE AF --TAGS_TO_REVERSE AF_EUR
--TAGS_TO_REVERSE AF_EAS --TAGS_TO_REVERSE AF_AMR
--TAGS_TO_REVERSE AF_SAS --TAGS_TO_REVERSE AF_AFR
--TAGS_TO_REVERSE AF_EUR_unrel --TAGS_TO_REVERSE AF_EAS_unrel
--TAGS_TO_REVERSE AF_AMR_unrel --TAGS_TO_REVERSE AF_SAS_unrel
--TAGS_TO_REVERSE AF_AFR_unrel --TAGS_TO_DROP AC
--TAGS_TO_DROP AC_EUR --TAGS_TO_DROP AC_EAS --TAGS_TO_DROP
AC_AMR --TAGS_TO_DROP AC_SAS --TAGS_TO_DROP AC_AFR
--TAGS_TO_DROP AC_EUR_unrel --TAGS_TO_DROP AC_EAS_unrel
--TAGS_TO_DROP AC_AMR_unrel --TAGS_TO_DROP AC_SAS_unrel
--TAGS_TO_DROP AC_AFR_unrel --TAGS_TO_DROP AC_Hom_EUR
--TAGS_TO_DROP AC_Hom_EAS --TAGS_TO_DROP AC_Hom_AMR
--TAGS_TO_DROP AC_Hom_SAS --TAGS_TO_DROP AC_Hom_AFR
--TAGS_TO_DROP AC_Hom --TAGS_TO_DROP AC_Het_EUR --TAGS_TO_DROP
AC_Het_EAS --TAGS_TO_DROP AC_Het_AMR --TAGS_TO_DROP AC_Het_SAS
--TAGS_TO_DROP AC_Het_AFR --TAGS_TO_DROP AC_Het --TAGS_TO_DROP
AC_Hom_EUR_unrel --TAGS_TO_DROP AC_Hom_EAS_unrel
--TAGS_TO_DROP AC_Hom_AMR_unrel --TAGS_TO_DROP
AC_Hom_SAS_unrel --TAGS_TO_DROP AC_Hom_AFR_unrel
--TAGS_TO_DROP AC_Het_EUR_unrel --TAGS_TO_DROP
AC_Het_EAS_unrel --TAGS_TO_DROP AC_Het_AMR_unrel
```

```
--TAGS_TO_DROP AC_Het_SAS_unrel --TAGS_TO_DROP
AC_Het_AFR_unrel
```

Finally, our bash scripts perform analyses and output three summary files, one for each tool (`analysis.CrossMap.txt`, `analysis.gatk.txt`, `analysis.Genozip.txt`), which describe the outcome of the variants (Lifted or Dropped) categorised from Genozip's oStatus categories. See Supplementary Information section 6 for the content of these files.

2.2.2 Indel benchmark results summary

Table S1. Performance of three liftover tools for indels, including handling of problematic variants, according to standard categories reported by the Genozip software.

# Variants	Category	Genozip	LiftoverVcf	CrossMap
18,201	Lifted without issues	✓ ***Lifted***	✓ ***Lifted***	✓ ***Lifted***
78	REF not mapped in chain file	✓ ***Unmapped***	✓ ***Unmapped***	✓ ***Unmapped***
153	REF⇄ALT switch but REF unchanged (switch in number of repeats)	✓ ***Lifted***	✗ ***Data Corruption***	✗ ***Data Corruption***
27	REF⇄ALT switch but REF unchanged (Flanking regions indicate switch)	✓ ***Lifted***	✗ ***Data Corruption***	✗ ***Data Corruption***
6	Simple Deletion->Insertion REF⇄ALT switch	✓ ***Lifted***	✗ ***Variant Loss***	✗ ***Data corruption***
71	Deletion->Insertion REF⇄ALT switch called by Deletion payload in chain file gap	✓ ***Lifted***	✗ ***Variant Loss***	✗ ***Variant Loss***
20	Deletion with payload partially in gap	✓ ***Unmapped***	✓ ***Unmapped***	✗ ***Data Corruption***
40	Insertion with reverse strand	✓ ***Lifted***	✗ ***Data Corruption (35)*** ✓ ***Lifted (5)***	✗ ***Data Corruption***
67	Deletion with reverse strand	✓ ***Lifted***	✗ ***Data Corruption (51)*** ✓ ***Lifted (16)***	✗ ***Data Corruption***
9	Apparent REF<>ALT switch disqualified by flanking regions	✗ ***Variant Loss***	✗ ***Data Corruption***	✗ ***Data Corruption***
13	REF changed in Deletion but not REF⇄ALT switch	✗ ***Variant Loss***	✗ ***Variant Loss***	✗ ***Data Corruption***
11	REF unchanged, but flanking regions show that this is a new Insertion allele.	✗ ***Variant Loss***	✗ ***Data Corruption***	✗ ***Data Corruption***

6	The REF base mismatches between the references	✗ ***Variant Loss***	✗ ***Variant Loss***	✗ ***Data Corruption***
4	REF unchanged, but flanking regions show that this is a new Deletion allele.	✗ ***Variant Loss***	✗ ***Data Corruption***	✗ ***Data Corruption***
18,706	**TOTAL**			

2.2.3 REF⇄ALT switch, entire variant mapped, no REF change

From Table S1:

# Variants	Genozip	LiftoverVcf	CrossMap
180	***Lifted***	***Data Corruption***	***Data Corruption***

Genozip oSTATUS: `OkRefAltSwitchIndelRpts`, `OkRefAltSwitchIndelFlank`

We found that both LiftoverVcf and CrossMap fail to identify 180 REF⇄ALT switches in indel variants, respectively, resulting in incorrect variants with the REF and ALT reversed vs their correct values, along with errors in many of the INFO and FORMAT annotations that depend on the order of alleles (such as AC, AF, AD, GL, etc.). Genozip lifts all 186 REF⇄ALT switches,

Case 1: simple insertion with REF⇄ALT switch

Consider this GRCh37 insertion variant, which maps to 16423118 in GRCh38:

```
#CHROM  POS       ID     REF    ALT
22      16903845  LN=133 T      TAC
```

```
> genocat --reference hs37d5.ref.genozip -r chr22:16903845+8
16903845-16903852       TAGAGGCA
```

```
> genocat --reference GRCh38.ref.genozip -r chr22:16423118+10
16423118-16423127       TACAGAGGCA
```

It is easy to see that the insertion got incorporated in the GRCh38 reference, therefore the haplotypes with GT=1 in the original VCF, indicating their samples have this insertion, are the REF allele in the Luft reference, while those haplotypes that don't have this insertion would be a deletion variant relative to the Luft reference. In other words, this is a REF⇄ALT switch.

Genozip correctly executes a REF⇄ALT switch (Table S2; note the fields that differ from the original VCF in red), while both LiftoverVcf and CrossMap fail to detect this REF⇄ALT switch and therefore generate a data-corrupted variant.

Table S2: Example VCF of an insertion with REF⇄ALT switch.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	genocat indel-37-38.d.vcf.genozip --luft --samples 1 --no-header -g LN=133	genocat indel-37-38.d.vcf.genozip --luft --samples 1 --no-header -g LN=133	grep -w LN=133 indel.38.gatk.vcf |cut -f1-10	grep -w LN=133 indel.38.CrossMap.vcf |cut -f1-10
#CHROM	22	**chr22**	**chr22**	22
POS	16903845	**16423118**	**16423118**	**16423118**
ID	LN=133	LN=133	LN=133	LN=133
REF	T	**TAC**	T	T
ALT	TAC	**T**	TAC	TAC
QUAL	166	166	166	166
FILTER	PASS	PASS	PASS	PASS
INFO	AA=TAC;ERATE=0.0118;RSQ=0.5469;AN=2184;LDAF=0.2750;VT=INDEL;THETA=0.0057;AVGPOST=0.7502;AC=455;AF=0.21;ASN_AF=0.14;AMR_AF=0.19;AFR_AF=0.44;EUR_AF=0.12;**LUFT=chr22,16423118,TAC,-**	AA=TAC;ERATE=0.0118;RSQ=0.5469;AN=2184;**LDAF=0.7250**;VT=INDEL;THETA=0.0057;AVGPOST=0.7502;**AC=1729**;**AF=0.79;ASN_AF=0.86;AMR_AF=0.81;AFR_AF=0.56;EUR_AF=0.88;PRIM=22,16903845,T,-**	AA=TAC;AC=455;AF=0.21;AFR_AF=0.44;AMR_AF=0.19;AN=2184;ASN_AF=0.14;AVGPOST=0.7502;ERATE=0.0118;EUR_AF=0.12;LDAF=0.2750;RSQ=0.5469;THETA=0.0057;VT=INDEL	AA=TAC;ERATE=0.0118;RSQ=0.5469;AN=2184;LDAF=0.2750;VT=INDEL;THETA=0.0057;AVGPOST=0.7502;AC=455;AF=0.21;ASN_AF=0.14;AMR_AF=0.19;AFR_AF=0.44;EUR_AF=0.12
FORMAT	GT:DS:GL	GT:DS:GL	GT:DS:GL	GT:DS:GL
HG00096	0|0:0.050:0.00,-1.20,-18.40	**1|1:1.950:-18.40,-1.20,0.00**	0|0:0.050:0.00,-1.20,-18.40	0|0:0.050:0.00,-1.20,-18.40

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF; highlight: errors.

Case 2: Insertion with repeats with REF⇄ALT switch

Consider the variant at POS=22780917 that maps to 22426580 on GRCh38:

```
#CHROM   POS        ID         REF      ALT
22       22780917   LN=3114    C        CA
```

Viewing the Primary (37) and Luft references: Notice that the Insertion was incorporated in the Luft reference—it has 3 As instead of 2:

```
> genocat -e hs37d5.ref.genozip -r chr22:22780917+9
22780917-22780925        CAATCGGTC

> genocat -e GRCh38.ref.genozip -r chr22:22426580+10
22426580-22426589        CAAATCGGTC
```

Genozip executed a REF⇄ALT switch while the other tools failed to do so (Table S3).

Table S3: Example VCF of an insertion with repeats with REF⇄ALT switch.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	genocat indel-37-38.d.vcf.genozip -H -s1 -g LN=3114	genocat indel-37-38.d.vcf.genozip -H -s1 --luft -g LN=3114	grep -w LN=3114 indel.38.gatk.vcf |cut -f1-10	grep -w LN=3114 indel.38.CrossMap.vcf |cut -f1-10
#CHROM	22	**chr22**	**chr22**	22
POS	22780917	**22426580**	**22426580**	**22426580**
ID	LN=3114	LN=3114	LN=3114	LN=3114
REF	C	**CA**	C	C
ALT	CA	**C**	CA	CA
QUAL	763	763	763	763
FILTER	PASS	PASS	PASS	PASS
INFO	AA=.;ERATE=0.0005;AN=2184;VT=IND	AA=.;ERATE=0.0005;AN=2184;VT=IND	AA=.;AC=1517;AF=0.69;AFR_AF=0.86	AA=.;ERATE=0.0005;AN=2184;VT=IND

	EL;THETA=0.0005;AC=1517;LDAF=0.6930;RSQ=0.9505;AVGPOST=0.9681;AF=0.69;ASN_AF=0.63;AMR_AF=0.56;AFR_AF=0.86;EUR_AF=0.71;**LUFT=chr22,22426580,CA,-**	EL;THETA=0.0005;**AC=667;LDAF=0.3070**;RSQ=0.9505;AVGPOST=0.9681;**AF=0.31;ASN_AF=0.37;AMR_AF=0.44;AFR_AF=0.14;EUR_AF=0.29;PRIM=22,22780917,C,-**	;AMR_AF=0.56;AN=2184;ASN_AF=0.63;AVGPOST=0.9681;ERATE=0.0005;EUR_AF=0.71;LDAF=0.6930;RSQ=0.9505;THETA=0.0005;VT=INDEL	EL;THETA=0.0005;AC=1517;LDAF=0.6930;RSQ=0.9505;AVGPOST=0.9681;AF=0.69;ASN_AF=0.63;AMR_AF=0.56;AFR_AF=0.86;EUR_AF=0.71
FORMAT	GT:DS:GL	GT:DS:GL	GT:DS:GL	GT:DS:GL
HG00096	0|1:1.000:-7.40,0.00,-6.20	**1|0:1.000:-6.20,0.00,-7.40**	0|1:1.000:-7.40,0.00,-6.20	0|1:1.000:-7.40,0.00,-6.20

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF; highlight: errors.

Case 3: Deletion with repeats with REF⇄ALT switch

Consider the variant at POS=24483878 that maps to 24087925 on GRCh38:

```
#CHROM  POS       ID       REF    ALT
22      24483878  LN=4150  AT     A
```

Viewing the Primary (37) and Luft (38) references: Notice that the Deletion was incorporated in the Luft reference - it has only 2 repeating Ts instead of 3:

```
> genocat -e hs37d5.ref.genozip -r chr22:24483878+10
24483878-24483887       ATTTAGGGAC

> genocat -e GRCh38.ref.genozip -r chr22:24087925+9
24087925-24087933       ATTAGGGAC
```

Genozip executed a REF⇄ALT switch while the other tools failed to do so (Table S4).

Table S4: Example VCF of a deletion with REF⇄ALT switch.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	genocat indel-37-38.d.vcf .genozip -H -s 1 -g LN=4150	genocat indel-37-38.d.vcf .genozip -H -s 1 --luft -g LN=4150	grep -w LN=4150 indel.38.gatk.vcf | cut -f1-10	grep -w LN=4150 indel.38.CrossMap. vcf | cut -f1-10
#CHROM	22	chr22	chr22	22
POS	24483878	24087925	24087925	24087925
ID	LN=4150	LN=4150	LN=4150	LN=4150
REF	AT	A	AT	AT
ALT	A	AT	A	A
QUAL	785	785	785	785
FILTER	PASS	PASS	PASS	PASS
INFO	AA=AT;AC=1441;AN=2184;LDAF=0.6585;VT=INDEL;THETA=0.0006;AVGPOST=0.99	AA=AT;AC=743;AN=2184;LDAF=0.3415;VT=INDEL;THETA=0.0006;AVGPOST=0.991	AA=AT;AC=1441;AF=0.66;AFR_AF=0.46;AMR_AF=0.82;AN=2184;ASN_AF=0.55;AV	AA=AT;AC=1441;AN=2184;LDAF=0.6585;VT=INDEL;THETA=0.0006;AVGPOST=0.9913;R

	13;RSQ=0.9854;ERATE=0.0006;AF=0.66;ASN_AF=0.55;AMR_AF=0.82;AFR_AF=0.46;EUR_AF=0.80;**LUFT=chr22,24087925,A,-**	3;RSQ=0.9854;ERATE=0.0006;**AF=0.34;ASN_AF=0.45;AMR_AF=0.18;AFR_AF=0.54;EUR_AF=0.20;PRIM=22,24483878,AT,-**	GPOST=0.9913;ERATE=0.0006;EUR_AF=0.80;LDAF=0.6585;RSQ=0.9854;THETA=0.0006;VT=INDEL	SQ=0.9854;ERATE=0.0006;AF=0.66;ASN_AF=0.55;AMR_AF=0.82;AFR_AF=0.46;EUR_AF=0.80
FORMAT	GT:DS:GL	GT:DS:GL	GT:DS:GL	GT:DS:GL
HG00096	0|1:1.000:-2.20,0.00,-15.80	**1|0:1.000:-15.80,0.00,-2.20**	0|1:1.000:-2.20,0.00,-15.80	0|1:1.000:-2.20,0.00,-15.80

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF; highlight: errors.

2.2.4 Deletion REF⇄ALT switch, REF change

From Table S1:

# Variants	Genozip	LiftoverVcf	CrossMap
6	***Lifted***	***Variant Loss***	***Data Corruption***

Genozip oSTATUS: `OkRefAltSwitchDelToIns`

We found that 6 variants with REF⇄ALT switch resulting in a REF change are either lost (LiftoverVcf) or mishandled leading to Data Corruption (CrossMap). Genozip successfully lifted these variants.

Consider the variant at POS=17998325 that maps to 17519295 on GRCh38:

```
#CHROM   POS        ID         REF      ALT
22       17998325   LN=825     TTG      T
```

Viewing the Primary (37) and Luft (38) references in the local context:

```
> genocat -e hs37d5.ref.genozip -r chr22:17998325+10
17998322-17998344       GTTTTGTTTTTTTTTTTTGAGAC

> genocat -e GRCh38.ref.genozip -r chr22:17519295 +10
17519292-17519314       GTTTTTTTTTTTTTTTTTGAGAC
```

The variant observed in the samples might be a true deletion or might indeed mirror the variation between the references, in which case it would have been better categorized as a SNP rather than a deletion. Regardless, given that this is categorized as a deletion in the VCF on hand, Genozip declares it a REF⇄ALT switch since the haplotypes with this variant become the REF allele in the Luft reference, and the ones without the variant are the ALT allele.

In cases like this where the REF changes between the references (here: TTG->TTT), LiftoverVcf rejects the variant with “MismatchedRefAllele”, whereas CrossMap simply updates the REF to TTT causing a Data Corruption (because the REF allele no longer represents the true sequences of the haplotypes with GT=0) (Table S5).

Table S5: Example VCF of a REF⇄ALT switch with REF change.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	genocat indel-37-38.d.vcf.genozip -H -s 1 -g LN=825	genocat indel-37-38.d.vcf.genozip -H -s 1 --luft -g LN=825	Variant lost	grep -w LN=825 indel.38.CrossMap.vcf | cut -f1-10
#CHROM	22	**chr22**		22
POS	17998325	**17519295**		**17519295**
ID	LN=825	LN=825		LN=825
REF	TTG	**T**		**TTT**
ALT	T	**TTG**		T
QUAL	191	191		191
FILTER	PASS	PASS		PASS
INFO	AA=.;AC=1970;AF=0.90;AFR_AF=0.99;AMR_AF=0.85;AN=2184;ASN_AF=0.96;AVGPOST=0.9826;ERATE=0.0014;EUR_AF=0.83;LDAF=0.8963;RSQ=0.9295;THETA=0.0003;VT=INDEL;LUFT=chr22,17519295,T,-	AA=.;**AC=214;AF=0.10;AFR_AF=0.01;AMR_AF=0.15;**AN=2184;**ASN_AF=0.04**;AVGPOST=0.9826;ERATE=0.0014;**EUR_AF=0.17;LDAF=0.1037**;RSQ=0.9295;THETA=0.0003;VT=INDEL**;PRIM=22,17998325,TTG,-**		AA=.;AC=1970;AF=0.90;AFR_AF=0.99;AMR_AF=0.85;AN=2184;ASN_AF=0.96;AVGPOST=0.9826;ERATE=0.0014;EUR_AF=0.83;LDAF=0.8963;RSQ=0.9295;THETA=0.0003;VT=INDEL
FORMAT	GT:DS:GL	GT:DS:GL		GT:DS:GL
HG00096	1|1:2.000:-9.00,-1.70,0.00	**0|0:0.000:0.00,-1.70,-9.00**		1|1:2.000:-9.00,-1.70,0.00

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF; highlight: errors.

2.2.5 Deletion REF⇄ALT switch with payload in gap

From Table S1:

# Variants	Genozip	LiftoverVcf	CrossMap
71	***Lifted***	***Variant Loss***	***Variant Loss***

Genozip oSTATUS: `OkRefAltSwitchWithGap`

When a deletion variant in the Primary reference enters the Luft reference, in other words, the deletion payload that existed in the Primary reference no longer exists in the Luft reference, this will often manifest itself as a gap in the chain file. We found 71 of these variants; lifting over with LiftoverVcf and CrossMap resulted in Variant Loss, while Genozip handled these variants correctly.

Consider for example:

```
#CHROM   POS        ID          REF          ALT
22       17995661   LN=821      TTTGCTGTTG   T
```

In the chain file we have the following alignments:

```
> genocat GRCh37_to_GRCh38.chain.genozip --show-chain
…
Primary: 22 17995313-17995661 Luft: chr22 17516283-17516631 Xstrand=-
Primary: 22 17995671-17996285 Luft: chr22 17516632-17517246 Xstrand=-
…
```

As can be appreciated, the anchor base of the variant, **`T`**, is the final base on the first alignment (that ends at 17995661), and the entire 9-base payload, **`TTGCTGTTG`**, precisely fits in the gap between the alignments (17995662 to 17995670).

When inspecting the two references starting at the anchor base **`T`**, it is clear that the Luft reference incorporates this deletion, and therefore the variant in a REF⇄ALT switch:

```
> genocat --reference hs37d5.ref.genozip -r chr22:17995661+15
17995661-17995675        TTTGCTGTTGTTGCC
```

```
> genocat -reference GRCh38.ref.genozip -r chr22:17516631+6
17516631-17516636        TTTGCC
```

These variants are correctly categorized as a REF⇄ALT switch by Genozip. However, LiftoverVcf rejects them with “NoTarget” and CrossMap rejects them with “Fail(REF==ALT)”, leading to Variant Loss (Table S6).

Table S6: Example VCF of a REF⇄ALT switch with payload in gap.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	genocat indel-37-38.d.vcf.genozip -r 17995661 -s HG00104 --header-one	genocat indel-37-38.d.vcf.genozip -r 17516631 -s HG00104 --luft	Variant lost	Variant lost
#CHROM	**22**	**chr22**		
POS	**17995661**	**17516631**		
ID	LN=821	LN=821		
REF	**TTTGCTGTTG**	**T**		
ALT	**T**	**TTTGCTGTTG**		
QUAL	1144	1144		
FILTER	PASS	PASS		
INFO	AA=TTTGCTGTTG;ERATE=0.0124;AN=2184;VT=INDEL;THETA=0.0005;AVGPOST=0.9345;**AC=1914**;**LDAF=0.8546**;RSQ=0.8066;**AF=0.88**;**ASN_AF=0.94**;**AMR_AF=0.84**;**AFR_AF=0.95**;**EUR_AF=0.80**;**LUFT=chr22,17516631,T,-**	AA=TTTGCTGTTG;ERATE=0.0124;AN=2184;VT=INDEL;THETA=0.0005;AVGPOST=0.9345;**AC=270**;**LDAF=0.1454**;RSQ=0.8066;**AF=0.12**;**ASN_AF=0.06**;**AMR_AF=0.16**;**AFR_AF=0.05**;**EUR_AF=0.20**;**PRIM=22,17995661,TTTGCTGTTG,-**		
FORMAT	GT:DS:GL	GT:DS:GL		
HG00104	**0|1:0.550:0.00,-1.20,-41.70**	**1|0:1.450:-41.70,-1.20,0.00**		

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF.

2.2.6 Deletion with payload partially in gap

From Table S1:

# Variants	Genozip	LiftoverVcf	CrossMap
20	***Variant Loss***	***Variant Loss***	***Data Corruption***

Genozip oSTATUS: `REFSplitInChain`

When a deletion payload is partially in a chain file gap (other than the case where the entire payload is in the gap), CrossMap updates the REF by removing the bases that fall in the gap. This is incorrect: while with the lifted REF the haplotypes that contain the variant (GT=1) are now correctly represented, the haplotypes with GT=0 are incorrectly represented as the actual sequenced data contains the previous REF, not the lifted one. Rather, this should be a new allele.

In contrast, since neither Genozip nor LiftoverVcf are capable of adding an allele they both correctly reject these variants (with the resulting Variant Loss).

Example: the indel test file:
```
#CHROM   POS        ID           REF       ALT
22       17995306   LN=818       ATTATAT   A
```

CrossMap-lifted variant:
```
#CHROM   POS        ID           REF       ALT
22       17516277   LN=818       ATTATA    A
```

The chain file has the last base in REF (POS=17995312) in the gap between two alignments:
```
Primary: chr22 17992953-17995311 Luft: chr22 17513924-17516282
Primary: chr22 17995313-17995661 Luft: chr22 17516283-17516631
```

2.2.7 Insertion with reverse strand

From Table S1:

# Variants	**Genozip**	**LiftoverVcf**	**CrossMap**
40	***Lifted***	***Data Corruption (35)*** ***Lifted (5)***	***Data Corruption***

Genozip oSTATUS: `OkRefSameInsRev`

Consider the following Insertion variant in our indel test file (in GRCh37 coordinates):

```
#CHROM   POS               ID       REF      ALT      INFO(partial)
22       16566319          LN=75    A        ACAAT    AC=13;AF=0.01;AN=2184
```

Per the chain file, POS maps to 15411644 in GRCh38, on the reverse strand. CrossMap lifts this insertion by reverse complementing the REF and ALT:

```
#CHROM   POS               ID       REF      ALT      INFO(partial)
chr22    15411644          LN=75    T        ATTGT    AC=13;AF=0.01;AN=2184
```

While the CrossMap-lifted variant contains precisely the same information as the original variant, it is unfortunately non-compliant with the VCF specification (section 5.2) that requires the variant's anchor base (**A** lifted to **T**) be on the left side, whereas here it is right-anchored. This is likely to break downstream tools.

LiftoverVcf goes further, and left-aligns the resulting variant, leading to this:

```
#CHROM   POS               ID       REF      ALT      INFO(partial)
chr22    15411637          LN=75    C        CTGAT    AC=13;AF=0.01;AN=2184
```

The algorithm LiftoverVcf applied here is as follows:
GRCh38 (Luft) in region 15411637-15411644 is **CTGATTGT**. Since **TGATTG** is 1½ repeats of the insertion variant payload **ATTG** (traversing backwards from the anchor base

T), it seems that we can represent this variant in a canonical left-aligned way with **CTGAT** at POS=15411637, because the original insertion without left-aligning **CTGATTG**^{ATTG}**T** yields precisely the same sequence as the insertion after left aligning: **C**^{TGAT}**TGATTGT**.

However, this is wrong. The reason is that there is no requirement in the VCF specification that a VCF file must contain all the variants of its specific samples, and it is not true that any loci lacking a variant is an indication that all the samples in the VCF file have a base equal to the reference at that locus. Only the loci covered by the variants listed in the VCF file are known, and we cannot make any assumption regarding the bases the specific samples in the VCF file have at other loci.

Consider, for example, the LiftoverVcf-generated left-aligned variant above. This variant now asserts that the 13 haplotypes (AC=13) with GT=1 for this variant, have the bases **CTGAT** starting at position 15411637. This is simply not knowable from the data at hand, and might, in fact, be wrong for any of the 13 haplotypes. For these 13 haplotypes in this VCF file, we know nothing at all about their bases in loci 15411637–15411643, all we know is that they have an insertion of **ATTG** just before locus 15411644.

This is therefore a risk of Data Corruption: if any one of the 13 haplotypes doesn't have **CTGAT** bases starting at 15311637, or if any one of the other 2171 haplotypes doesn't have a **C** at this locus, then the VCF file is corrupted.

In addition to the above issue of Data Corruption, there is an additional issue of Data Loss: the original VCF informed us that all 2184 haplotypes have a **T** at 15411644, and that 13 haplotypes have a **ATTG** just before 15411644. This information is no longer present in the LiftoverVcf-generated file, and therefore lost.

Genozip chooses to left-anchor the variant, but not left-align it:

```
#CHROM   POS              ID       REF      ALT      INFO(partial)
chr22    15411643         LN=75    G        GATTG    AC=13;AF=0.01;AN=2184
```

This is quite similar to the CrossMap variant, except Genozip's anchor base **G** is to the left, rather than the right, of the insertion payload **`ATTG`**, which is also reflected in POS. This makes it compliant with the VCF specification, while not risking Data Corruption.

Note that this solution is still not perfect: namely, it asserts that all samples in the VCF have **G** at 15411643, which is not known from the data, and it loses the information that all samples have a **T** at 15411644. However, without access to the full sequence of all samples, or alternatively knowledge that the VCF contains all variants in the samples (and taking into account neighboring variants in the computation), this is the best that can be done. We consider this issue to be a *Minor data loss* that does not affect the categorization.

All 40 variants with reverse strand Insertion were affected for CrossMap, but only 33 of them for LiftoverVcf - the remaining 7 were cases where left-aligning resulted in the same variant as left-anchoring.

An additional Data Corruption issue is that if an INFO/AA field exists, both CrossMap and LiftoverVcf fail to update it appropriately. This issue affects 14 of the 40 variants, including 2 of the 7 for which LiftoverVcf correctly left-anchored. See the example in Table S7.

Finally, we note a *Minor Data Loss* in LiftoverVcf is due to conversion of the FORMAT/GL field to FORMAT/PL, with the loss of granularity. As discussed, a *Minor Data Loss* does not affect the categorization.

Table S7: Example VCF of insertion with reverse strand.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	`genocat indel-37-38.d.vcf .genozip -H -s 1 -g LN=107`	`genocat indel-37-38.d.vcf. genozip -H -s 1 --luft -g LN=107`	`grep -w LN=107 indel.38.gatk.vcf | cut -f1-10`	`grep -w LN=107 indel.38.CrossMap .vcf | cut -f1-10`
#CHROM	22	**chr22**	**chr22**	22
POS	16687501	**15290461**	**15290461**	**15290462**
ID	LN=107	LN=107	LN=107	LN=107
REF	C	**C**	**C**	**G**
ALT	CA	**CT**	**CT**	**TG**
QUAL	208	208	208	208
FILTER	PASS	PASS	PASS	PASS
INFO	AA=CA;AC=103;AF=0.05;AFR_AF=0.07;AMR_AF=0.04;AN=2184;ASN_AF=0.05;AVGPOST=0.9793;ERATE=0.0006;EUR_AF=0.03;LDAF=0.0530;RSQ=0.8577;THETA=0.0152;VT=INDEL;**LUFT=chr22,15290461,C,X**	**AA=CT**;AC=103;AF=0.05;AFR_AF=0.07;AMR_AF=0.04;AN=2184;ASN_AF=0.05;AVGPOST=0.9793;ERATE=0.0006;EUR_AF=0.03;LDAF=0.0530;RSQ=0.8577;THETA=0.0152;VT=INDEL;**PRIM=22,16687501,C,X**	AA=CA;AC=103;AF=0.05;AFR_AF=0.07;AMR_AF=0.04;AN=2184;ASN_AF=0.05;AVGPOST=0.9793;ERATE=0.0006;EUR_AF=0.03;LDAF=0.0530;RSQ=0.8577;**ReverseComplementedAlleles**;THETA=0.0152;VT=INDEL	AA=CA;AC=103;AF=0.05;AFR_AF=0.07;AMR_AF=0.04;AN=2184;ASN_AF=0.05;AVGPOST=0.9793;ERATE=0.0006;EUR_AF=0.03;LDAF=0.0530;RSQ=0.8577;THETA=0.0152;VT=INDEL
FORMAT	GT:DS:GL	GT:DS:GL	GT:DS:**PL**	GT:DS:GL
HG00096	0|0:0.000:0.00,-0.60,-8.40	0|0:0.000:0.00,-0.60,-8.40	0|0:0.000:**0,6,84**	0|0:0.000:0.00,-0.60,-8.40
			0|1:1.050:48,3,0	0|1:1.050:-4.80,-0.30,0.00

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF; highlight: errors.

2.2.8 Deletion with reverse strand

From Table S1:

# Variants	**Genozip**	**LiftoverVcf**	**CrossMap**
67	***Lifted***	***Data Corruption (51)*** ***Lifted (16)***	***Data Corruption***

Genozip oSTATUS: `OkRefSameDelRev`

This issue is similar to the previous one, just for deletions rather than insertions.

Consider the following deletion, in GRCh37 coordinates:

```
#CHROM   POS         ID       REF      ALT     INFO(partial)
22       16524572    LN=70    ACACT    A       AC=112;AF=0.05;AN=2184
```

CrossMap lifts it by reverse-complementing REF and ALT and moving POS from what became REF's right-most base, 15453391, to its left-most base 15453387:

```
#CHROM   POS         ID       REF      ALT     INFO(partial)
chr22    15453387    LN=70    AGTGT    T       AC=112;AF=0.05;AN=2184
```

As discussed for insertions, this variant contains correct information, however it is non-compliant with the VCF specification, and hence we will consider it a Data Corruption.

LiftoverVcf, as in insertions, goes further, and left-aligns the resulting variant, creating the following variant:

```
#CHROM  POS         ID       REF      ALT     INFO(partial)
chr22    15453384   LN=70    CTGAG    C       AC=112;AF=0.05;AN=2184
```

Brief explanation: GRCh38, chr22, region 15453384-15453391 is: **CTGAGTGT**. The deletion generated by the original (CrossMap) variant **CTG**$_{\mathbf{AGTG}}$**T** results in an identical sequence as the deletion described by the LiftoverVcf's left-aligned variant: **C**$_{\text{TGAG}}$**TGT**. However, as before, this is wrong because it makes possibly incorrect assumptions about the nucleotide sequences of the samples at loci 15453384–15453386 which are not in fact knowable from the data.

Again, similar to the insertion case, Genozip left-anchors but does not left-align the variant, which is the optimal (yet still imperfect) solution:

```
#CHROM   POS         ID       REF      ALT     INFO(partial)
chr22    15453386    LN=70    GAGTG    G       AC=112;AF=0.05;AN=2184
```

All 67 variants with reverse strand deletion were affected for CrossMap, but only 46 of them for LiftoverVcf—the remaining 21 were cases where left-aligning resulted in the same variant as left-anchoring, however, 5 of the 21 are nevertheless corrupted due to failure to update the INFO/AA field, bringing the total of corrupted LiftoverVcf variants to 51.

2.2.9 REF changed but not REF⇄ALT switch

From Table S1:

# Variants	**Genozip**	**LiftoverVcf**	**CrossMap**
19	***Variant Loss***	***Variant Loss***	***Data Corruption***

Genozip oSTATUS: `RefNewAlleleDelRefChanged`, `RefNewAllelInsRefChanged`

In indels, if the bases of the REF change between the two references, CrossMap simply updates the REF field with the new bases. This is completely wrong, and is responsible for corruption of 26 variants in our test file. In contrast, both Genozip and LiftoverVcf reject the variants in this case (losing their data).

Example 1 (Insertion):

Consider the following insertion variant in our indel test file (in GRCh37 coordinates):

```
#CHROM  POS       ID      REF    ALT    INFO(partial)
22      18068419  LN=871  A      ATT    AC=311;AF=0.14;AN=2184
```

POS=18068419 in GRCh37 maps to 17585653 in GRCh38, and this position has a base change—from A in GRCh37 to T in GRCh38. Because of this base change, both Genozip and LiftoverVcf reject this variant. However, CrossMap produced the following:

```
#CHROM  POS       ID      REF    ALT    INFO(partial)
22      17585653  LN=871  T      ATT    AC=311;AF=0.14;AN=2184
```

This is obviously completely wrong. First, it changed a left-anchored insertion (with an A anchor base) to a right-anchored insertion (with a T anchor base). Second, it is asserting that all 311 haplotypes (AC=311) with GT=1 have a T at this position, while in fact the original VCF informs us that they have an A. Finally, it is an invalid insertion variant per the VCF specification 5.2.

Example 2 (Deletion):

Consider the following deletion variant in our indel test file (in GRCh37 coordinates):

```
#CHROM  POS       ID      REF    ALT    INFO(partial)
22      18068421  LN=872  TA     T      AC=631;AF=0.29;AN=2184
```

The CrossMap lifted variant:

```
#CHROM  POS       ID      REF    ALT    INFO(partial)
22      17585655  LN=872  TT     T      AC=631;AF=0.29;AN=2184
```

The reference has a base change in the second base of REF. Therefore, this would be a new allele and the correct variant would be REF=TT ALT=TA,T. Since neither Genozip nor LiftoverVcf are capable of adding an allele, they both reject this variant. However, CrossMap's variant asserts that all the (AN-AC)=1553 haplotypes with GT=0 have a TT at this position, while in fact they have a TA.

2.2.10 New allele when REF unchanged

From Table S1:

# Variants	**Genozip**	**LiftoverVcf**	**CrossMap**
15	***Variant Loss***	***Data Corruption***	***Data Corruption***

Genozip oSTATUS: `RefNewAlleleInsSameRef`, `RefNewAlleleDelSameRef`

Cases where the bases of REF are identical in both references, yet the new reference contains a new allele. These variants are lifted incorrectly by both CrossMap and LiftoverVcf yielding corrupted variants. In contrast, Genozip rejects them because it is not capable of adding an allele.

Example:

Original variant:

```
#CHROM   POS        ID          REF      ALT
22       22735735   LN=3091     T        TG
```

CrossMap and LiftoverVcf variant in GRCh38—unchanged REF and ALT:

```
#CHROM   POS        ID          REF      ALT
chr22    22381366   LN=3091     T        TG
```

Looking at the references:

```
> genocat -e hs37d5.ref.genozip -r chr22:22735735+10
22735735-22735744       TAGGGAACTG


> genocat -e GRCh38.ref.genozip -r chr22:22381366+10
22381366-22381375       TGGGGAACTG
```

At first glance, this might look like a REF⇄ALT switch since TG is present in the Luft reference. However, looking at the variant in its local context, it is clear that the Luft reference represents a new allele that is neither REF nor ALT, and the new variant would be:

```
#CHROM   POS        ID        REF      ALT
chr22    22381366   LN=3091   TG       TA,TAG
```

Genozip correctly identifies this as a case of a new allele, and rejects the variant as it is not capable of adding another allele. CrossMap and LiftoverVcf in contrast, incorrectly lift the variant resulting in Data Corruption.

2.2.11 New allele - not quite a REF⇄ALT switch

From Table S1:

# Variants	Genozip	LiftoverVcf	CrossMap
9	***Variant Loss***	***Data Corruption***	***Data Corruption***

Genozip oSTATUS: `RefNewAlleleIndelNoSwitch`

Consider the following case:

```
#CHROM   POS        ID        REF     ALT
22       22484247   LN=2870   A       AC
```

Inspecting both references, with 4 flanking bases, we see:

Primary reference: `CAGGAAATG`

Luft reference: `CAGGACAGTG`

As first glance, it appears to be a REF⇄ALT switch where the insertion AC got incorporated in the Luft reference. However, Genozip also compares the flanking 4 bases on either side to verify that is indeed a REF⇄ALT switch. In this case, the region to the right is different - an AATG in the Primary reference vs AGTG in the Luft reference, and therefore Genozip rejects the REF⇄ALT switch hypothesis and instead determines that the Luft reference represents a new allele that is neither the REF nor the ALT. Since Genozip cannot currently add new alleles, it rejects this variant with `RefNewAlleleIndelNoSwitch`.

We also experimented with a value of 2 for the length of the flanking regions to be tested. When this more permissive approach is used, we observed the following changes in the oSTATUS categories of the variants in the indel test file (Table S8).

Table S8: Variant categorisation changes if changing the flanking regions test from four bases on either side, to two.

oSTATUS	**4 bases**	**2 bases**
RefNewAlleleIndelNoSwitch	9	2
OkRefAltSwitchIndelFlank	27	34
RefNewAlleleDelRefChanged	13	10
RefNewAlleleInsSameRef	17	11
OkRefAltSwitchDelToIns	6	9
RefNewAllelInsRefChanged	0	6
Other categories - no change		

2.3 SNP benchmark

2.3.1 Data preparation

We developed a set of scripts that executes this benchmark in its entirely - from downloading the data to producing the analysis files. It is available from https://github.com/divonlan/genozip-dvcf-results and the entry point script is: run-snp-37-38.sh

The key steps executed by this script are these:

The SNP test file, SS6004478.annotated.nh2.variants.vcf.gz, containing 4,109,729 SNP variants, was extracted from the tar archive: https://sharehost.hms.harvard.edu/genetics/reich_lab/sgdp/vcf_variants/vcfs.variants.public_samples.279samples.tar.

We used the same reference files and chain file as the indel test.

Similar to the indel test, we proceed to preparing the DVCF:

```
> genozip --chain GRCh37_to_GRCh38.chain.genozip
SS6004478.annotated.nh2.variants.vcf.gz --add-line-numbers
--match-chrom-to-reference -o snp-37-38.d.vcf.genozip
```

We then proceed to generate a GRCh37-only file with contains the add lines numbers, the updated contig names and adding the INFO/oSTATUS field to each variant, reporting Genozip's lift-over status:

```
> genocat snp-37-38.d.vcf.genozip --single --show-ostatus -o
snp.37.annotated.vcf
```

Testing CrossMap: As in the indel test, CrossMap.py version v0.5.2 was used:

```
> CrossMap.py vcf GRCh37_to_GRCh38.chain.gz snp.37.annotated.vcf
GRCh38_full_analysis_set_plus_decoy_hla.fa snp.38.CrossMap.vcf
```

Testing GATK LiftoverVCF: As in the indel test, LiftoverVcf contained in GATK version 4.17 was used to lift over the snp test file to GRCh38:

```
> gatk --java-options '-Xmx16g -XX:ParallelGCThreads=1' LiftoverVcf
--INPUT snp.37.annotated.vcf --OUTPUT snp.38.gatk.vcf --CHAIN
GRCh37_to_GRCh38.matched.chain --REJECT snp.38.gatk.rejects.vcf
--RECOVER_SWAPPED_REF_ALT --REFERENCE_SEQUENCE
GRCh38_full_analysis_set_plus_decoy_hla.fa.gz --TAGS_TO_REVERSE AF
--TAGS_TO_REVERSE MLEAF --TAGS_TO_DROP AC --TAGS_TO_DROP MLEAC
```

As in the indel test, our bash script produces three files: `analysis.CrossMap.txt` `analysis.gatk.txt` `analysis.Genozip.txt`.

2.3.2 SNP benchmark results summary

Table S9. Performance of three liftover tools for SNPs, including handling of problematic variants, according to standard categories reported by the Genozip software. Note that LiftoverVcf contains command line options that determine the handling of REF⇄ALT switches. Variants can either be i) dropped (the default, which result in Variant Loss), ii) kept, with updated annotations for a subset of variants that LiftoverVcf is able to revise (resulting in Data Corruption due to a subset of variants maintaining incorrect annotations), or iii) kept, with some annotations being converted or dropped (resulting in Annotation Loss). We chose the latter option in our benchmarks, which we consider to be the least problematic.

# Vars.	**Category**	**Genozip**	**LiftoverVcf**	**CrossMap**
4,037,520	No issues - lifted	✓***Lifted***	✓***Lifted***	✓***Lifted***
26,728	No issues - no mapping	✓***Unmapped***	✓***Unmapped***	✓***Unmapped***
29,635	REF⇄ALT switch in SNPs	✓***Lifted***	✗ ***Annotation Loss***	✗ ***Variant Loss***
15,689	Annotation update upon strand reversal	✓***Lifted***	✗ ***Data Corruption***	✗ ***Data Corruption***
12	IUPACs	✓***Lifted***	✗ ***Variant Loss***	✗ ***Data Corruption***
68	REF change, not to ALT, in bi-allelic SNPs when AF<1	✗ ***Variant Loss***	✗ ***Variant Loss***	✗ ***Data Corruption***
30	REF change in multi-allelic SNPs when AF<1	✗ ***Variant Loss***	✗ ***Variant Loss***	✗ ***Data Corruption***
47	REF change, not to ALT, in bi-allelic SNPs when AF=1	✓***Lifted***	✗ ***Variant Loss***	✓***Lifted***
4,109,729	**TOTAL**			

2.3.3 REF⇄ALT switch in SNPs

From Table S9:

# Variants	Genozip	LiftoverVcf	CrossMap
29635	***Lifted***	***Annotation Loss***	***Variant Loss***

Genozip oSTATUS: `OkRefAltSwitchSNP`

In case of a REF⇄ALT switch in a bi-allelic SNP, Genozip lifts the variant, updating all the relevant annotations. CrossMap drops all these variants. LiftoverVcf drops these variants by default, but is capable of lifting them with the --RECOVER_SWAPPED_REF_ALT option, however this it offers very limited corrections (with --TAGS_TO_REVERSE and --TAGS_TO_DROP), which would cause data loss (if fields are dropped) or corruption (if they are not).

In the following example (Table S10), we used LiftoverVcf options

```
--RECOVER_SWAPPED_REF_ALT --TAGS_TO_REVERSE AF
--TAGS_TO_REVERSE MLEAF.
```

LiftoverVcf correctly updates the fields INFO/AF, INFO/MLEAF and FORMAT/GT, FORMAT/AD, FORMAT/PL but fails to correct INFO/AC, INFO/MLEAC. We can avoid a Data Corruption due to AC and MLEAC by also applying --TAGS_TO_DROP AC and --TAGS_TO_DROP MLEAC, and hence we categorize LiftoverVcf as Annotation Loss for these variants.

We also note that LiftoverVcf has an additional *Minor Data Loss* due to re-ordering of the FORMAT annotations, thereby losing the information of their original order.

Table S10: Example VCF of REF⇄ALT switch in a SNP.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	`genocat snp-37-38.d.vcf.genozip -H -s 1 -g LN=632`	`genocat snp-37-38.d.vcf.genozip -H -s 1 --luft -g LN=632`	`grep -w LN=632 snp.38.gatk.vcf | cut -f1-10`	
#CHROM	1	**chr1**	**chr1**	
POS	770568	**835188**	**835188**	
ID	LN=632	LN=632	LN=632	
REF	A	**G**	**G**	
ALT	G	**A**	**A**	
QUAL	809.01	809.01	809.01	
FILTER	.	.	**PASS**	
INFO	AC=2;AF=1.00;AN=2;BaseCounts=0,0,31,1;DB;DP=32;Dels=0.00;FS=0.000;GC=47.13;HaplotypeScore=0.0000;MLEAC=2;MLEAF=1.00;MQ=32.04;MQ0=4;QD=25.28;**LUFT=chr1,835188,G,-**	**AC=0;AF=0.00**;AN=2;BaseCounts=0,0,31,1;DB;DP=32;Dels=0.00;FS=0.000;GC=47.13;HaplotypeScore=0.0000;**MLEAC=0;MLEAF=0.00**;MQ=32.04;MQ0=4;QD=25.28;**PRIM=1,770568,A,-**	**AF=0.00**;AN=2;BaseCounts=0,0,31,1;DB;DP=32;Dels=0.00;FS=0.000;GC=47.13;HaplotypeScore=0.0000;**MLEAF=0.00**;MQ=32.04;MQ0=4;QD=25.28;**SwappedAlleles**	
FORMAT	GT:AD:DP:GQ:PL:FL	GT:AD:DP:GQ:PL:FL	GT:AD:DP:FL:GQ:PL	
SS6004478	1/1:0,31:31:63:809,63,0:N	**0/0:31,0**:31:63:**0,63,809**:N	**0/0:31,0**:31:N:63:**0,63,809**	

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF; highlight: errors.

2.3.4 Annotation update upon strand reversal

From Table S9:

# Variants	**Genozip**	**LiftoverVcf**	**CrossMap**
15689	***Lifted***	***Data Corruption***	***Data Corruption***

Genozip oSTATUS: `OkRefSameSNPRev`

Sometimes when REF, ALT are reverse-complemented due to the chain file mapping being to the reverse strand, it is necessary to update some annotations. Both LiftoverVcf and CrossMap fail to do so, resulting in data-corrupted variants. In our test file, the affected annotation is INFO/BaseCounts annotation (Table S11).

Table S11: Example VCF of annotation update upon strand reversal.

	Genozip (37)	**Genozip (38)**	**LiftoverVcf (38)**	**CrossMap (38)**
	genocat snp-37-38.d.vcf.genozip -H -s 1 -g LN=253	genocat snp-37-38.d.vcf.genozip -H -s 1 --luft -g LN=253	grep -w LN=253 snp.38.gatk.vcf | cut -f1-10	grep -w LN=253 snp.38.CrossMap.vcf | cut -f1-10
#CHROM	1	**chr1**	**chr1**	1
POS	364127	**455210**	**455210**	**455210**
ID	LN=253	LN=253	LN=253	LN=253
REF	G	**C**	**C**	**C**
ALT	A	**T**	**T**	**T**
QUAL	26.78	26.78	26.78	26.78
FILTER	.	.	PASS	.
INFO	AC=2;AF=1.00;AN=2;BaseCounts=5,0,23,0;BaseQRankSum=0.804;DB;DP=28;Dels=0.00;FS=0.000;GC=38.65;HaplotypeScore=0.0000;MLEAC=2;MLEAF=1.00;MQ=4.61;MQ0=26;MQRankSum=0.804;QD=0.96;ReadPosRankSum=0.804;**LUFT=chr1,455210,C,X**	AC=2;AF=1.00;AN=2;**BaseCounts=0,23,0,5**;BaseQRankSum=0.804;DB;DP=28;Dels=0.00;FS=0.000;GC=38.65;HaplotypeScore=0.0000;MLEAC=2;MLEAF=1.00;MQ=4.61;MQ0=26;MQRankSum=0.804;QD=0.96;ReadPosRankSum=0.804;**PRIM=1,364127,G,X**	AC=2;AF=1.00;AN=2;BaseCounts=5,0,23,0;BaseQRankSum=0.804;DB;DP=28;Dels=0.00;FS=0.000;GC=38.65;HaplotypeScore=0.0000;MLEAC=2;MLEAF=1.00;MQ=4.61;MQ0=26;MQRankSum=0.804;QD=0.96;ReadPosRankSum=0.804;**ReverseComplementedAlleles**	AC=2;AF=1.00;AN=2;BaseCounts=5,0,23,0;BaseQRankSum=0.804;DB;DP=28;Dels=0.00;FS=0.000;GC=38.65;HaplotypeScore=0.0000;MLEAC=2;MLEAF=1.00;MQ=4.61;MQ0=26;MQRankSum=0.804;QD=0.96;ReadPosRankSum=0.804
FORMAT	GT:AD:DP:GQ:PL:FL	GT:AD:DP:GQ:PL:FL	GT:AD:DP:FL:GQ:PL	GT:AD:DP:GQ:PL:FL
SS6004478	1/1:23,5:27:3:25,30:N	1/1:23,5:27:3:25,30:N	1/1:23,5:27:N:3:25,30	1/1:23,5:27:3:25,3,0:N

GRCh version is in parentheses in the headers. The VCF lines are presented transposed, for readability. **Red**: changes vs. the original VCF; highlight: errors.

2.3.5 IUPACs

From Table S9:

# Variants	Genozip	LiftoverVcf	CrossMap
12	***Lifted***	***Variant Loss***	***Data Corruption***

Genozip oSTATUS: `OkRefSameSNPIupac`

The SNP test file has 12 variants at loci that contain a non-ACTGN IUPAC base in GRCh38. All 12 variants have a REF that is a base that is included in the mapped IUPAC base in GRCh38. For example (17, 81077361, T) is mapped to (chr17, 83129591, W). W is defined as A or T.

Since T is included in W, Genozip calls this variant as `OkRefSameSNP` and lifts it. LiftoverVcf rejects this variant because T≠W, which is a valid call but yet an unfortunate loss of data.

CrossMap on the other hand, replaces the REF with the IUPAC base, thereby generating variants that not only contain less information than the original variant (as the haplotypes with GT=0 had a definite base as specified by the original REF, not an ambiguous one) and hence represent a Data Loss, but are also noncompliant with the VCF 4.3 specification (violation of requirement 1.6.1-REF: "Each base must be one of A,C,G,T,N") and hence are likely to break downstream analysis tools:

```
#CHROM   POS         ID           REF    ALT
13       100973393   LN=3041971   K      T
13       100973395   LN=3041972   Y      T
17       83128871    LN=3550982   K      T
17       83128888    LN=3550984   Y      T
17       83129591    LN=3550985   W      A
17       83130798    LN=3550987   Y      T
17       83130998    LN=3550988   Y      T
17       83131245    LN=3550989   R      A
17       83131933    LN=3550990   Y      T
```

17	83133010	LN=3550991	R	A
17	83133390	LN=3550993	Y	T
17	83133686	LN=3550994	Y	T

2.3.6 REF change, not to ALT, in bi-allelic SNPs when AF<1

From Table S9:

# Variants	Genozip	LiftoverVcf	CrossMap
68	***Variant Loss***	***Variant Loss***	***Data Corruption***

Genozip oSTATUS: `RefNewAlleleSNP`

When the REF base of a SNP changes between the references, and unless this is a REF⇄ALT switch in a bi-allelic SNP, CrossMap simply updates the new REF without updating ALT. This is correct only in the case where there are no haplotypes with the REF allele (i.e.

$\sum_{alt} AC_{alt} = AN$ or $\sum_{alt} AF_{alt} = 1$).

Example:

Original VCF (GRCh37):

```
#CHROM   POS        ID         REF     ALT     INFO(partial)
1        13808732   LN=20854   C       T       AC=1;AF=0.500;AN=2
```

CrossMap incorrectly-lifted VCF (GRCh38):

```
#CHROM   POS        ID         REF     ALT     INFO(partial)
1        13482278   LN=20854   G       T       AC=1;AF=0.500;AN=2
```

Since G is a new allele the correct lifting should have been:

```
#CHROM   POS        ID         REF     ALT     INFO(partial)
1        13482278   LN=20854   G       T,C     AC=1,1;AF=0.5,0.5;AN=2
```

Genozip and LiftoverVcf, in contrast, reject these variants as they cannot handle adding an allele.

2.3.7 REF change in multi-allelic SNPs when AF<1

From Table S9:

# Variants	Genozip	LiftoverVcf	CrossMap
30	***Variant Loss***	***Variant Loss***	***Data Corruption***

Genozip oSTATUS: `RefMultiAltSwitchSNP`

In cases of multi-allelic SNPs with a reference base change, CrossMap changes REF without updating ALT, regardless of whether the new reference is one of the ALT alleles.

Example:

Original VCF (GRCh37):

```
#CHROM  POS       ID         REF     ALT  INFO(partial)
18      77831522  LN=3663131 G       C,T  AC=1,1;AF=0.500,0.500;AN=2
```

CrossMap incorrectly-lifted VCF (GRCh38):

```
#CHROM  POS       ID         REF     ALT  INFO(partial)
18      80073165  LN=3663131 C       C,T  AC=1,1;AF=0.500,0.500;AN=2
```

This correct lifting would have been a REF⇄ALT switch:

```
#CHROM  POS       ID         REF     ALT  INFO(partial)
18      80073165  LN=3663131 C       G,T  AC=0,1;AF=0,0.500;AN=2
```

Genozip and LiftoverVcf, in contrast, reject these variants as they cannot handle REF changes in multi-allelic SNPs.

2.3.8 REF change, not to ALT, in bi-allelic SNPs when AF=1

From Table S9:

# Variants	Genozip	LiftoverVcf	CrossMap
47	***Lifted***	***Variant Loss***	***Lifted***

Genozip oSTATUS: `OkNewRefSNP`

In these bi-allelic SNP variants, there is a reference base change, and the sample doesn't contain any haplotypes with the REF allele, i.e. AC=AN or AF=1. Therefore, it is permissible to just update the REF (the old REF that would normally become one of the ALT alleles, is redundant in this case since it has AF=0). However, LiftoverVcf fails to do so, needlessly rejecting these variants.

Example:

Original VCF (GRCh37):

```
#CHROM   POS        ID          REF      ALT   INFO(partial)
1        99597759   LN=137361   C        G     AC=2;AF=1.00;AN=2
```

Genozip and CrossMap correctly-lifted VCF (GRCh38)—REF replacement OK if AF=1

```
#CHROM   POS        ID           REF      ALT   INFO(partial)
chr1     99132203   LN=137361    A        G     AC=2;AF=1.00;AN=2
```

2.3.9 REF⇄ALT switch proportions

We now turn our attention to the 0.7% (29,635 out of 4,109,729) of the variants in our test file that are categorized as REF⇄ALT switches. These variants are dropped by CrossMap and in LiftoverVcf, they are either dropped or some of their annotations are dropped, depending on the command line options used. In contrast. Genozip lifts them over correctly, updating the annotations that sensitive to REF⇄ALT switches (see https://genozip.com/dvcf-rendering.html)

Here we show that despite the overall number of these variants being relatively small, they are not uniformly distributed across the genome but rather preferentially located in certain regions, and therefore dropping them may introduce bias in certain downstream analyses, such as GWAS or selection scans.

We divided the genome into 30,749 *windows* of 100 kb each, and counted for each 100-kb window the total number of variants in the SNP test file within the window vs. the number of those variants that are a REF⇄ALT switch. This was done by leveraging Genozip's internal genome-wide position called GPOS (for Global POSition). The position is that of the base as it appears in the original FASTA file used to generate the reference file, when all contigs are concatenated in the order they appear in the FASTA.
The first command, below, outputs the number of variants per 100-kb window in the test file. The first column in the output is the count of variants in a 100-kb window and the second is the sequential GPOS in units of 100k (the first line missing a number is window 0). For brevity, we show here the first 6 lines.

```
> genocat snp-37-38.d.vcf.genozip -e hs37d5.ref.genozip --gpos
-HG | cut -f2 | rev | cut -c6- | rev | uniq -c
    113
     60 1
     79 2
      4 3
      6 4
    124 5
```

The second command, below, outputs the number of variants with REF⇄ALT switch per 100-kb window in the test file. The first column of the output is the count of variants with REF⇄ALT switch and the second is the sequential GPOS in units of 100k. For brevity, we show here first 6 regions that have any REF⇄ALT switch variants at all):

```
> genocat snp-37-38.d.vcf.genozip -e hs37d5.ref.genozip --gpos
-HG --show-dvcf --grep OkRefAltSwitchSNP | cut -f4 | rev |cut
-c6- | rev | uniq -c
      1 7
      2 8
      1 11
      1 12
      1 13
     22 15
```

The proportion of REF⇄ALT switch variants compared to the total number of variants for each 100-kb window of our SNP test file are shown in Figure 1C. It is easy to see that distribution of REF⇄ALT switch variants across the genome is highly non-uniform – the vast majority of the 100kb windows have very few REF⇄ALT switch variants, while a small number of windows have a very high percentage of REF⇄ALT switch variants – some in which over 80% of the variants are switches. Therefore, when CrossMap or LiftoverVcf drop all variants with a REF⇄ALT switch, they are potentially introducing bias to the data that might impact downstream analyses. Another view of the same data is presented in Figure S3.

We then proceeded to compare the GRCh38 coordinates of REF⇄ALT switch variants to the regions of the GRCh38 reference genome with known issues, downloaded from https://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/Issue_Mapping/GRCh38.p13_issues.gff3. Indeed, there is significant overlap between the loci with REF⇄ALT switches and regions of the reference genome known to be problematic, see Figure S4. The R script used for this analysis and generate the Figure S4 is available from https://github.com/divonlan/genozip-dvcf-results/tree/main/Fig-2A.

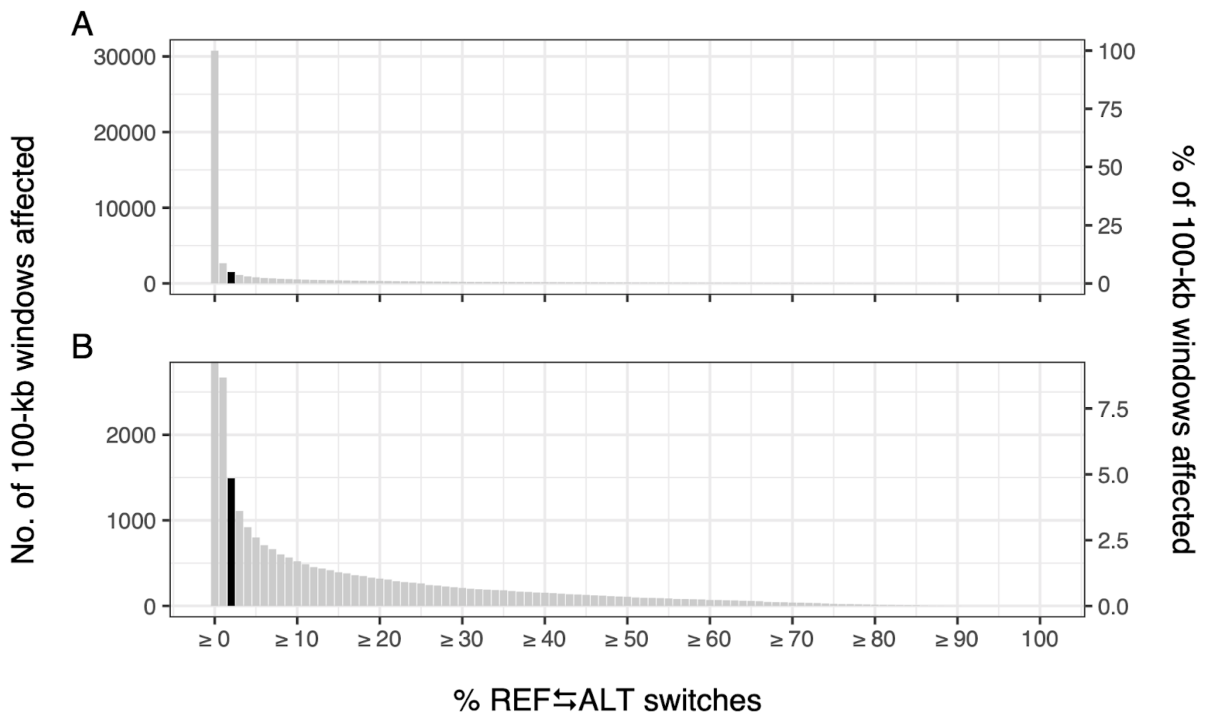


Figure S3. **Distribution of the number of 100-kb windows with at least *x*% REF⇄ALT switch variants**. A. The number of 100-kb windows was calculated for increments of 1% of REF⇄ALT switch variants content. The y-axis on the right side of the figure indicates the corresponding percentage of affected windows relative to all 100-kb windows. B. A close up of the distribution showing that nearly 5% (black bar) of all 100-kb windows in the human genome contain at least 2% REF⇄ALT switch variants amongst all variants in the window.

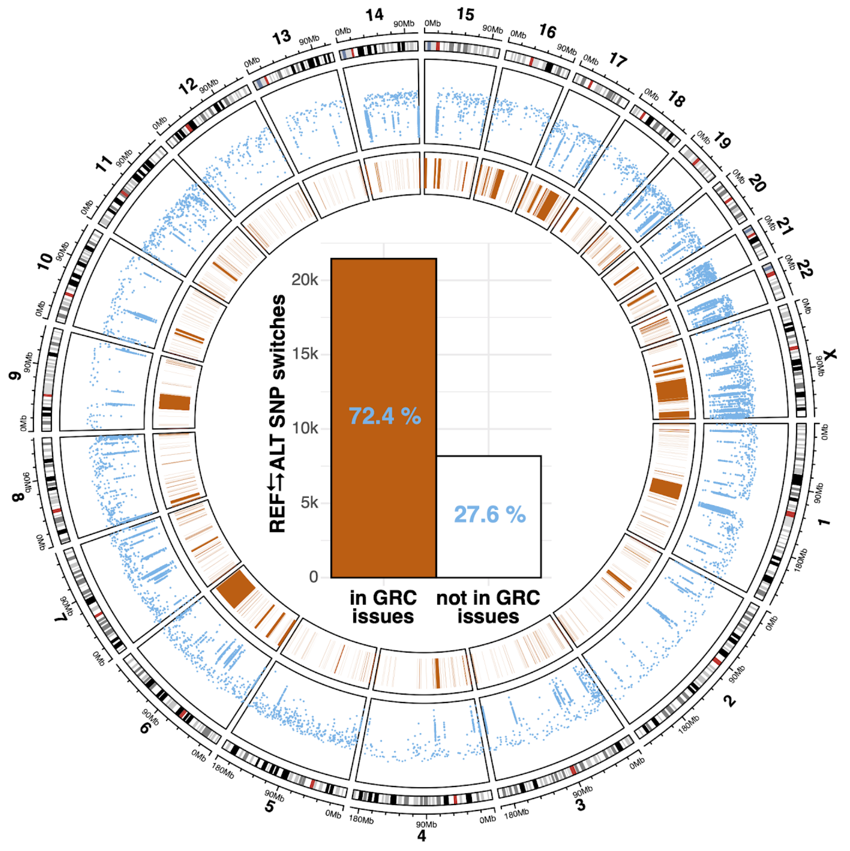


Figure S4. Distribution and functional impact of REF⇄ALT allele switches in SNP variants. Circos plot: the location of REF⇄ALT allele switches are shown in the blue rainfall plot, with GRC-identified problematic regions shown as orange polygons. Bar plot: Number (bars) and percentage (blue text) of REF⇄ALT allele switches inside or outside problematic GRC regions.

2.4 Benchmark summary

Table S12: Summary of correctly (in green: Lifted or Unmapped) vs incorrectly (in red: Data Corruption or Variant Loss) lifted variants for each tested tool. For each lifting tool, we show the number of variants and the percentage of variants falling under each major category, including outcomes that could negatively impact downstream analyses (i.e., Data Corruption).

	Indels						***SNPs***					
	Genozip		***LiftoverVcf***		***CrossMap***		***Genozip***		***LiftoverVcf***		***CrossMap***	
Total	*18706*	*100%*	*18706*	*100%*	*18706*	*100%*	*4109729*	*100%*	*4109729*	*100%*	*4109729*	*100%*
Correct	*18663*	*99.8%*	*18304*	*97.9%*	*18279*	*97.7%*	*4109631*	*99.998%*	*4064248*	*98.9%*	*4064295*	*98.9%*
Correct that are lifted over	*18565*	*99.2%*	*18206*	*97.3%*	*18201*	*97.3%*	*4082903*	*99.3%*	*4037520*	*98.2%*	*4037567*	*98.2%*
Incorrect	*43*	*0.2%*	*402*	*2.1%*	*427*	*2.3%*	*98*	*0.002%*	*45481*	*1.1%*	*45434*	*1.1%*
Incorrect that lead to Data Corruption	*0*	*0%*	*306*	*1.6%*	*356*	*1.9%*	*0*	*0%*	*15689*	*0.4%*	*15799*	*0.4%*
Incorrect that are REF⇄ALT switches	*0*	*0%*	*257*	*1.3%*	*257*	*1.3%*	*0*	*0%*	*29635*	*0.7%*	*29635*	*0.7%*

SI.3. ClinVar analysis

3.1 Data preparation

We analysed the ClinVar file from the week of 03 Jan 2022 downloaded from https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh37/weekly/clinvar_20220103.vcf.gz

The analysis can be run by executing the bash script `run-clinvar-37-38.sh` in the github repository https://github.com/divonlan/genozip-dvcf-results. The key parts of this script are:

Lifting the file to a DVCF:

```
> genozip --echo --chain
shared/GRCh37_to_GRCh38.matched.chain.genozip --add-line-numbers
--match-chrom-to-reference shared/clinvar.37.vcf.gz -o
clinvar-37-38/clinvar-37-38.d.vcf.genozip
```

As before, to allow easy detection of variants with potential issues, we created a ClinVar GRCh37 single-coordinate file that contains an extra `INFO/oSTATUS` field, as well as the line numbers in the ID field:

```
> genocat clinvar-37-38/clinvar-37-38.d.vcf.genozip --single -o
clinvar-37-38/clinvar.37.annotated.vcf --show-ostatus
```

For CrossMap, we used the following command:

```
> CrossMap.py vcf shared/GRCh37_to_GRCh38.matched.chain
clinvar-37-38/clinvar.37.annotated.vcf
shared/GRCh38_full_analysis_set_plus_decoy_hla.fa.gz
clinvar-37-38/clinvar.38.CrossMap.vcf
```

For LiftoverVcf, we used the following command:

```
> gatk --java-options '-Xmx16g -XX:ParallelGCThreads=1' LiftoverVcf
--INPUT clinvar-37-38/clinvar.37.annotated.vcf --OUTPUT
clinvar-37-38/clinvar.38.gatk.vcf --CHAIN
shared/GRCh37_to_GRCh38.matched.chain --REJECT
clinvar-37-38/clinvar.38.gatk.rejects.vcf --RECOVER_SWAPPED_REF_ALT
--REFERENCE_SEQUENCE
shared/GRCh38_full_analysis_set_plus_decoy_hla.fa.gz
--TAGS_TO_REVERSE AF_ESP --TAGS_TO_REVERSE AF_EXAC --TAGS_TO_REVERSE
AF_TGP --TAGS_TO_DROP DUMMY
```

3.2 Genozip analysis

Genozip has no variants with corrupted data, however it did drop 201 variants (133 SNPs, 68 non-SNPs) which have a valid mapping in the chain file, because the allele represented by the GRCh38 reference is neither the REF nor the ALT allele.

Table S13: ClinVar variants dropped by Genozip despite having a mapping in the chain file. This occurs because the allele in the GRCh38 (Luft) reference is a new allele, neither REF nor ALT. This does not include variants correctly dropped due to lack of mapping in the chain file.

Type	#	Issue	ALLELEID	CLNSIG
Data loss	201	Allele in GRCh38 is a new allele (neither REF nor ALT)	696043,799115,685731,15474,964796,1170574,696055,696058,1167865,581994,696139,15681,696267,696378,696404,447571,425007,392854,697512,286118,451010,286231,390508,221328,708391,1154227,980680,963007,173441,697258,1154060,698145,452738,453319,215271,33220,455066,698872,174523,54118,1037727,1037728,226789,700218,389801,710823,961851,699897,699899,699900,699901,699902,699903,711517,790800,1155979,459012,959873,700419,700425,700426,700428,700429,700431,700436,700482,1008398,1156206,851221,174261,174548,700865,700867,1162096,800875,1162094,1162095,487407,549956,700869,700871,700872,692808,701351,397737,622275,701488,47952,240741,53156,701260,20181,167056,712775,390514,33351,838853,1156807,1166021,167084,917805,244624,150504,1156610,712655,702499,175552,175851,702467,713776,461651,702807,702808,703029,167190,529374,703257,703335,703373,242133,1157890,465467,818455,468332,1033832,343827,789384,1157991,376201,375315,706355,704364,704365,715791,55368,705368,245748,245880,245942,964061,536982,176732,705090,431872,334260,797925,682788,963904,11	135 Benign 17 Uncertain significance 15 Likely benign 10 Pathogenic 6 Likely pathogenic 6 Benign/Likely benign 3 not provided 3 association 2 risk factor 2 Conflicting interpretations of pathogenicity 1 other 1 drug response

			69844,882486,882487,40517,1158756,684879,684881,1158835,705475,705477,680034,966169,705738,963022,390511,390657,390590,390591,963023,705823,439050,38485,31934,47980,706250,706252,26660,706257,706259,706260,706263,706277,243824,963026,706035,670963,792118,706116,549821,670737,706133,706162,706163,706164,706168,706169,706170,706171,982298,472107,706177,706179,706180	

3.3 CrossMap analysis

Of the 969,410 variants in the file, 967,781 were lifted, and 1628 failed to lift. Some of the lifted variants were corrupt, and some of the dropped variants were unnecessarily dropped.

Table S14: ClinVar variants incorrectly dropped by CrossMap (excluding variants correctly dropped due to lack of mapping in the chain file), and variants lifted incorrectly.

Type	#	Issue	ALLELEID	CLNSIG
Data loss	204	Failed to lift REF⇄ALT switch	177885,191721,389423,106000,1163377,1163378,862175,1153237,353057,227743,1164070,1153536,249770,655075,1153962,389508,177658,141535,102135,389649,390440,670179,671101,291712,291961,789736,193757,389612,251402,1168025,167900,251996,36716,177851,54119,1037729,141976,141771,178204,141770,390476,390568,227764,389793,18435,699898,683010,1155956,684027,662663,662667,136076,1171823,304611,390475,313496,1156147,1156207,174544,54805,700866,800873,1171929,143080,654535,1171932,790945,701352,701350,701349,701348,701347,270003,142604,53180,389879,389938,254076,175704,254276,389897,17210,137366,701599,140343,54569,54553,254491,54551,55703,1157066,190700,254805,791396,1157289,230590,1157471,339619,323041,873957,656305,433550,390078,132224,192275,1157931,1158130,256553,344888,1087130,345325,1173004,791785,269599,506140,1163638,1173059,329799,256613,390307,791872,508906,344604,230979,257300,1168415,433895,142630,231042,257199,334255,177983,1173294,169607,344231,349365,178100,45542,353525,137338,257355,257356,257358,23429,716929,1158836,351613,1169888,106610,1164570,1164571,1164572,1164573,792021,817902,508193,817919,817923,817930,508196,818000,818032,818037,818051,818053,818054,818055,8180	160 Benign 21 drug response 9 Likely benign 8 Benign/Likely benign 3 Conflicting interpretations of pathogenicity 2 Uncertain significance 1 Pathogenic

			56,818057,818058,818060,818090,818094,818115,818122,717806,1159714,1159720,1159723,101448,792471,352903,134697,99042,98295,52521,800324,52519,656729,24924,670821,669859,25543,99678,101226,45455,352797,1159494,671190,352799,99399,25405,339086,99598	

Table S14: (continued from previous page)

Type	#	Issue	ALLELEID	CLNSIG
Data Corruption	201	Lifted has bad REF field	696043,799115,685731,15474,964796,1170574,696055,696058,1167865,581994,696139,15681,696267,696378,696404,447571,425007,392854,697512,286118,451010,286231,390508,221328,708391,1154227,980680,963007,173441,697258,1154060,698145,452738,453319,215271,33220,455066,698872,174523,54118,1037727,1037728,226789,700218,389801,710823,961851,699897,699899,699900,699901,699902,699903,711517,790800,1155979,459012,959873,700419,700425,700426,700428,700429,700431,700436,700482,1008398,1156206,851221,174261,174548,700865,700867,1162096,800875,1162094,1162095,487407,549956,700869,700871,700872,692808,701351,397737,622275,701488,47952,240741,53156,701260,20181,167056,712775,390514,33351,838853,1156807,1166021,167084,917805,244624,150504,1156610,712655,702499,175552,175851,702467,713776,461651,702807,702808,703029,167190,529374,703257,703335,703373,242133,1157890,465467,818455,468332,1033832,343827,789384,1157991,376201,375315,706355,704364,704365,715791,55368,705368,245748,245880,245942,964061,536982,176732,705090,431872,334260,797925,682788,963904,1169844,882486,882487,40517,1158756,684879,684881,1158835,705475,705477,680034,966169,705738,963022,390511,390657,390590,390591,963023,705823,439050,38485,31934,47980,706250,706252,26660,706257,706259,706260,706263,706277,243824,963026,706035,670963,792118,706116,549821,670737,706133,706162,706163,706164,706168,706169,706170,706171,982298,472107,706177,706179,706180	135 Benign 17 Uncertain significance 15 Likely benign 10 Pathogenic 6 Likely pathogenic 6 Benign/Likely benign 3 not provided 3 association 2 risk factor 1 other 1 drug response 2 Conflicting interpretations of pathogenicity

Data Corruption	4	Lifted failed to switch REF and ALT	776998,657462,198367,438998	3 Benign 1 Likely benign
Data Corruption	4	Mapped despite no mapping in chain file	1097855,407586,1156794,1157524	2 Likely benign 2 Benign

3.4 LiftoverVcf analysis

Of the 969,410 variants in the file, 967,816 were lifted, and 1594 failed to lift. Some of the lifted variants were corrupt, and some of the dropped variants were unnecessarily dropped:

Table S15: ClinVar variants incorrectly dropped by LiftoverVcf (excluding variants correctly dropped due to lack of mapping in the chain file), and variants lifted incorrectly.

Type	#	Issue	ALLELEID	CLNSIG
Data loss	162	Allele in GRCh38 is a new allele (neither REF nor ALT)	696043,685731,15474,964796,1170574,696055,696058,581994,696139,15681,696267,696378,696404,447571,425007,392854,697512,451010,708391,173441,697258,698145,452738,453319,33220,455066,698872,174523,54118,1037728,226789,700218,389801,710823,699897,699899,699900,699901,699902,699903,711517,459012,959873,700419,700425,700426,700428,700429,700431,700436,700482,1008398,1156206,851221,174261,174548,700865,700867,1162096,800875,1162094,1162095,700869,700871,700872,692808,701351,397737,622275,701488,47952,240741,53156,701260,20181,167056,712775,33351,838853,1166021,167084,917805,150504,712655,702499,175552,175851,702467,713776,702807,702808,167190,529374,703257,703335,703373,242133,465467,818455,468332,1033832,343827,789384,376201,375315,706355,704364,704365,715791,705368,245748,245880,245942,536982,705090,431872,334260,797925,682788,963904,882486,882487,40517,684879,684881,705475,705477,680034,966169,705738,963022,390591,705823,38485,31934,47980,706250,706252,26660,706257,706259,706260,706263,706277,243824,963026,706035,706116,549821,706133,706162,706163,706164,706168,706169,706170,706171,982298,472107,706177,706179,706180	107 Benign 13 Uncertain significance 11 Likely benign 9 Pathogenic 5 Likely pathogenic 5 Benign/Likely benign 3 not provided 3 association 2 risk factor 2 Conflicting interpretations of pathogenicity 1 other 1 drug response
Data loss	4	Failed to lift REF⇄ALT switch of non-SNPs	1037729,178204,683010,800324,16	2 Uncertain significance 2 Benign

Data corruption	39	Incorrect REF field (GRCh38 is a new allele)	964796,581994,1037728,389801,959873,851221,167056,167084,167190,818455,789384,245748,245880,245942,536982,431872,797925,963904,966169,963022,26660,963026	28 Benign 4 Uncertain significance 4 Likely benign 1 Pathogenic 1 Likely pathogenic 1 Benign/Likely benign
Data corruption	4	Lifted failed to switch REF and ALT	1037729,683010	3 Benign 1 Likely benign

3.5 ClinVar benchmark – summary

All tools dropped variants of “Pathogenic” clinical significance, leading to the conclusion that lift over techniques should never be used in a clinical setting. However, CrossMap and LiftoverVcf did worse than that – they also *incorrectly* lifted 10 (CrossMap) or 1 (LiftoverVcf) variants of “Pathogenic” clinical significance, resulting in a corrupt REF field of these variants, that could potentially lead to incorrect clinical diagnosis.

SI.4. GRCh38 and Telomere-to-Telomere

We repeated the tests described in section 1 and 3, but this time between the GRCh38 and Telomere-to-Telomere v1.0 reference.

- For the SNP test file, we used the GRCh38 file which was the output of the Genozip lift-over from GRCh37 to GRCh38.

- For the indels test file, we extracted the indel variants from a GRCh38 version of the 1000 Genome Project. This is an independent analysis of the 1KGP data, and *not* a lift-over of the GRCh37 file we used for testing. The GRCh38 file contains significantly more indel variants. The file was obtained from: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000G_2504_high_coverage/working/20201028_3202_raw_GT_with_annot/

- For the ClinVar test file, we obtained the GRCh38 version of the same data used in section 3, from: ftp://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/weekly/clinvar_20210724.vcf.gz

- The GRCh38 to T2Tv1.0 chain file was obtained from: http://t2t.gi.ucsc.edu/chm13/hub/t2t-chm13-v1.0/hg38Lastz/hg38.t2t-chm13-v1.0.over.chain.gz

- The T2Tv1.0 reference file was obtained from: https://s3-us-west-2.amazonaws.com/human-pangenomics/T2T/CHM13/assemblies/chm13.draft_v1.0.fasta.gz

- The GRCh38 reference, was obtained from: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/GRCh38_reference_genome/GRCh38_full_analysis_set_plus_decoy_hla.fa

As before, we provide bash scripts which execute the entire analysis including downloading the files. These scripts also serve as precise instructions for reproducing our results. They can be found in https://github.com/divonlan/genozip-dvcf-results.

We observe that the mapping of GRCh38 to T2T is significantly more complex than the mapping of GRCh37 to GRCh38, by the measure of the number of unique alignments in their respective chain files:

```
> genocat --show-chain GRCh37_to_GRCh38.matched.chain.genozip | wc -l
18389
```

```
> genocat --show-chain hg38.t2t-chm13-v1.0.over.matched.chain.genozip|wc -l
865173
```

A mapping of how each of the three tools handles the various variant categories appears in Tables S18, S19 and S20. We note that the tools diverge significantly with regards to how they handle the various cases. Note that “lifted” merely means that the variant appears in the output file—not that it is correct. In fact, there are many known cases in which the variant is incorrect—these are documented in Sections 2.2 and 2.3 and a mapping to the relevant section appears in the second column of each table. We also noted some new categories that did not appear in the GRCh37-to-GRCh38 case; these are marked as “New”. We did not perform a detailed analysis of the causes for the differences between the tools in this case of GRCh38-to-T2T liftover, which should be the purpose of future research.

Table S18. Categorization of indel variants and how each tool handles problematic variants.

Genozip oStatus	Section	Count	Genozip	LiftoverVcf	CrossMap
OkRefSameIndel	N/A	155,695	Lifted	Lifted	Lifted
OkRefSameNotLeftAnc	N/A	30,654	Lifted	30123 Lifted 531 Dropped	Lifted
OkRefSameDelRev	2.2.8	3,823	Lifted	3535 Lifted 288 Dropped	Lifted
OkRefSameInsRev	2.2.7	2,010	Lifted	1998 Lifted 12 Dropped	Lifted
OkRefAltSwitchIndelRpts	2.2.3	1,738	Lifted	1735 Lifted 3 Dropped	Lifted
OkRefSameNDNIRev	New	1,575	Lifted	1271 Lifted 304 Dropped	Lifted
OkRefAltSwitchWithGap	2.2.5	871	Lifted	Variant dropped	Variant dropped
OkRefAltSwitchIndelFlank	2.2.3	389	Lifted	Lifted	Lifted
RefSplitInChain	2.2.6	12,147	Dropped	Dropped	Lifted 2554 Dropped 9593
RefNotMappedInChain	N/A	5,688	Dropped	Dropped	Lifted 2127 Dropped 3561
RefMultiAltSwitchIndel	New	3,002	Dropped	Lifted 1695 Dropped 1307	Lifted
RefNewAlleleNotLeftAnc	New	2,835	Dropped	Lifted 53 Dropped 2782	Lifted
RefNewAlleleNDNI	New	2,459	Dropped	Lifted 44 Dropped 2415	Lifted
RefNewAlleleDelRefChanged	2.2.9	1,429	Dropped	Lifted 66 Dropped 1363	Lifted
RefNewAlleleInsSameRef	2.2.10	1,181	Dropped	Lifted	Lifted
RefNewAlleleDelSameRef	2.2.10	512	Dropped	Lifted 475 Dropped 37	Lifted
RefNewAlleleIndelNoSwitch	2.2.11	380	Dropped	Lifted	Lifted
RefNewAllelInsRefChanged	2.2.9	313	Dropped	Lifted 121 Dropped 192	Lifted
INFO/AF	New	167	Dropped	Dropped	Lifted
Total		**226,868**			

Table S19. Categorization of SNP variants and how each tool handles problematic variants.

Genozip oStatus	Section	Count	Genozip	LiftoverVcf	CrossMap
OkRefSameSNP	N/A	2,288,044	Lifted	Lifted	Lifted
OkRefAltSwitchSNP	2.3.3	1,705,586	Lifted	INFO/AC dropped	Dropped
OkRefSameSNPRev	2.3.4	22,266	Lifted	Lifted	Lifted
OkNewRefSNP	2.3.8	1,869	Lifted	Dropped	Lifted
NoMappingInChainFile	N/A	61,771	Dropped	Dropped	Dropped
RefNewAlleleSNP	2.3.6	1,934	Dropped	Dropped	Lifted
RefMultiAltSwitchSNP	2.3.7	1,433	Dropped	Dropped	Lifted
Total		**4,060,637**			

Table S20. Categorization of ClinVar variants and how each tool handles problematic variants.

Genozip oStatus	Section	Count	Genozip	LiftoverVcf	CrossMap
OkRefSameSNP	N/A	857,612	Lifted	Lifted	Lifted
OkRefSameIndel	N/A	86,095	Lifted	Lifted	Lifted
OkRefAltSwitchSNP	2.3.3	11,055	Lifted	Lifted	Dropped
OkRefSameNotLeftAnc	N/A	5,972	Lifted	Lifted	Lifted
OkRefSameSNPRev	2.3.4	2,058	Lifted	Lifted	Lifted
OkRefAltSwitchIndelRpts	2.2.3	734	Lifted	Lifted	Lifted
OkRefAltSwitchWithGap	2.2.5	432	Lifted	Dropped	Dropped 431 Lifted 1
OkRefAltSwitchIndelFlank	2.2.3	152	Lifted	Lifted	Lifted
OkRefSameDelRev	2.2.8	89	Lifted	Lifted 87 Dropped 2	Lifted
OkRefAltSwitchDelToIns	2.2.4	53	Lifted	Dropped	Lifted
OkRefAltSwitchNotLeftAnc	New	50	Lifted	Dropped	Dropped 34 Lifted 16
OkRefSameInsRev	2.2.7	48	Lifted	Lifted	Lifted
RefNewAlleleInsSameRef	2.2.10	1,071	Dropped	Lifted	Lifted
RefSplitInChain	2.2.6	1,003	Dropped	Dropped	Dropped 897 Lifted 106
RefNotMappedInChain	N/A	859	Dropped	Dropped	Dropped 791 Lifted 68
RefNewAlleleSNP	2.3.6	732	Dropped	Dropped	Lifted
RefNewAlleleDelRefChanged	2.2.9	520	Dropped	Dropped	Lifted
RefNewAlleleNotLeftAnc	New	457	Dropped	Dropped	Lifted 457 Dropped 1
RefNewAlleleDelSameRef	2.2.10	340	Dropped	Lifted	Lifted
RefNewAllelInsRefChanged	2.2.9	111	Dropped	Dropped 109 Lifted 2	Lifted
RefNewAlleleIndelNoSwitch	2.2.11	42	Dropped	Lifted	Lifted
ChromNotInPrimReference	N/A	1	Dropped	Dropped	Dropped
Total		**969,486**			

SI.Appendix. Analysis script output

Our analysis scripts in the https://github.com/divonlan/genozip-dvcf-results repository generate analysis output files for each tool. The Genozip analysis file summarizes the number of variants assigned to each category, while the analysis files for CrossMap and LiftoverVcf summarize the number of variants in each Genozip category, that were either lifted or dropped by the tool.

We therefore have analysis files for each of the three tools (Genozip, CrossMap and LiftoverVcf), for each of the three tests (Indels, SNPs and ClinVar) for each of the two datasets (GRCh37->GRCh38 and GRCh38->T2T):

1) Indels – GRCh37 to GRCh38

```
Genozip
Showing counts of o$TATUS (did_i=17). Total items=18706 Number of
categories=14
OkRefSameIndel	18201	97.30%
OkRefAltSwitchIndelRpts	153	0.82%
RefNotMappedInChain	78	0.42%
OkRefAltSwitchWithGap	71	0.38%
OkRefSameDelRev	67	0.36%
OkRefSameInsRev	40	0.21%
OkRefAltSwitchIndelFlank	27	0.14%
RefSplitInChain	20	0.11%
RefNewAlleleDelRefChanged	13	0.07%
RefNewAlleleInsSameRef	11	0.06%
RefNewAlleleIndelNoSwitch	9	0.05%
OkRefAltSwitchDelToIns	6	0.03%
RefNewAlleleInsRefChanged	6	0.03%
RefNewAlleleDelSameRef	4	0.02%

CrossMap
data=indel primary=37 luft=38 tool=CrossMap
Lifted OkRefSameIndel: 18201
Lifted OkRefAltSwitchIndelRpts: 153
Lifted RefNotMappedInChain: 2
Failed RefNotMappedInChain: 76
Failed OkRefAltSwitchWithGap: 71
Lifted OkRefSameDelRev: 67
Lifted OkRefSameInsRev: 40
Lifted OkRefAltSwitchIndelFlank: 27
Lifted RefSplitInChain: 18
Failed RefSplitInChain: 2
Lifted RefNewAlleleDelRefChanged: 13
Lifted RefNewAlleleInsSameRef: 11
Lifted RefNewAlleleIndelNoSwitch: 9
Lifted OkRefAltSwitchDelToIns: 6
Lifted RefNewAlleleInsRefChanged: 6
Lifted RefNewAlleleDelSameRef: 4

LiftoverVcf
data=indel primary=37 luft=38 tool=gatk
Lifted OkRefSameIndel: 18201
Lifted OkRefAltSwitchIndelRpts: 153
Failed RefNotMappedInChain: 78
Failed OkRefAltSwitchWithGap: 71
Lifted OkRefSameDelRev: 67
Lifted OkRefSameInsRev: 40
Lifted OkRefAltSwitchIndelFlank: 27
Failed RefSplitInChain: 20
Failed RefNewAlleleDelRefChanged: 13
Lifted RefNewAlleleInsSameRef: 11
Lifted RefNewAlleleIndelNoSwitch: 9
Failed OkRefAltSwitchDelToIns: 6
Failed RefNewAlleleInsRefChanged: 6
Lifted RefNewAlleleDelSameRef: 4
```

2) SNPs – GRCh37 to GRCh38

```
Genozip
Showing counts of o$TATUS (did_i=17). Total items=4109729 Number of
categories=8
OkRefSameSNP     4037520    98.24%
OkRefAltSwitchSNP     29635 0.72%
RefNotMappedInChain   26728 0.65%
OkRefSameSNPRev  15689 0.38%
RefNewAlleleSNP  68    0.00%
OkNewRefSNP      47    0.00%
RefMultiAltSwitchSNP  30    0.00%
OkRefSameSNPIupac     12    0.00%

CrossMap
data=snp primary=37 luft=38 tool=CrossMap
Lifted OkRefSameSNP: 4037520
Failed OkRefAltSwitchSNP: 29635
Failed RefNotMappedInChain: 26728
Lifted OkRefSameSNPRev: 15689
Lifted RefNewAlleleSNP: 68
Lifted OkNewRefSNP: 47
Lifted RefMultiAltSwitchSNP: 30
Lifted OkRefSameSNPIupac: 12

LiftoverVcf
data=snp primary=37 luft=38 tool=gatk
Lifted OkRefSameSNP: 4037520
Lifted OkRefAltSwitchSNP: 29635
Failed RefNotMappedInChain: 26728
Lifted OkRefSameSNPRev: 15689
Failed RefNewAlleleSNP: 68
Failed OkNewRefSNP: 47
Failed RefMultiAltSwitchSNP: 30
Failed OkRefSameSNPIupac: 12
```

3) ClinVar – GRCh37 to GRCh38

```
Genozip
Showing counts of o$TATUS (did_i=17). Total items=969410 Number of
categories=19
OkRefSameSNP    870016    89.75%
OkRefSameIndel  90450 9.33%
OkRefSameNotLeftAnc   6557  0.68%
RefNotMappedInChain   1424  0.15%
OkRefSameSNPRev 501   0.05%
OkRefAltSwitchSNP     200   0.02%
RefNewAlleleSNP 133   0.01%
RefNewAlleleInsSameRef      36    0.00%
OkRefSameDelRev 33    0.00%
RefNewAlleleDelRefChanged   20    0.00%
OkRefSameInsRev 16    0.00%
RefNewAlleleNotLeftAnc      7     0.00%
OkRefAltSwitchIndelRpts     4     0.00%
RefSplitInChain 4     0.00%
OkRefAltSwitchWithGap 2     0.00%
OkRefAltSwitchNotLeftAnc    2     0.00%
RefNewAlleleInsRefChanged   2     0.00%
RefNewAlleleIndelNoSwitch   2     0.00%
RefNewAlleleDelSameRef      1     0.00%

CrossMap
data=clinvar primary=37 luft=38 tool=CrossMap
Lifted OkRefSameSNP: 870016
Lifted OkRefSameIndel: 90450
Lifted OkRefSameNotLeftAnc: 6557
Lifted RefNotMappedInChain: 2
Failed RefNotMappedInChain: 1422
Lifted OkRefSameSNPRev: 501
Failed OkRefAltSwitchSNP: 200
Lifted RefNewAlleleSNP: 133
Lifted RefNewAlleleInsSameRef: 36
Lifted OkRefSameDelRev: 33
Lifted RefNewAlleleDelRefChanged: 20
Lifted OkRefSameInsRev: 16
Lifted RefNewAlleleNotLeftAnc: 7
Lifted OkRefAltSwitchIndelRpts: 4
Lifted RefSplitInChain: 2
Failed RefSplitInChain: 2
Failed OkRefAltSwitchWithGap: 2
Failed OkRefAltSwitchNotLeftAnc: 2
Lifted RefNewAlleleInsRefChanged: 2
Lifted RefNewAlleleIndelNoSwitch: 2
Lifted RefNewAlleleDelSameRef: 1

LiftoverVcf
data=clinvar primary=37 luft=38 tool=gatk
Lifted OkRefSameSNP: 870016
Lifted OkRefSameIndel: 90450
Lifted OkRefSameNotLeftAnc: 6557
Failed RefNotMappedInChain: 1424
Lifted OkRefSameSNPRev: 501
Lifted OkRefAltSwitchSNP: 200
Failed RefNewAlleleSNP: 133
Lifted RefNewAlleleInsSameRef: 36
Lifted OkRefSameDelRev: 33
Failed RefNewAlleleDelRefChanged: 20
Lifted OkRefSameInsRev: 16
Failed RefNewAlleleNotLeftAnc: 7
Lifted OkRefAltSwitchIndelRpts: 4
Failed RefSplitInChain: 4
Failed OkRefAltSwitchWithGap: 2
Failed OkRefAltSwitchNotLeftAnc: 2
Failed RefNewAlleleInsRefChanged: 2
Lifted RefNewAlleleIndelNoSwitch: 2
Lifted RefNewAlleleDelSameRef: 1
```

4) Indels – GRCh38 to T2T

```
Genozip
Showing counts of o$TATUS (did_i=17). Total items=227096 Number of
categories=20
OkRefSameIndel  155695     68.56%
OkRefSameNotLeftAnc   30654 13.50%
RefSplitInChain  12147 5.35%
RefNotMappedInChain   5688  2.50%
OkRefSameDelRev  3823  1.68%
RefMultiAltSwitchIndel      3002  1.32%
RefNewAlleleNotLeftAnc      2835  1.25%
RefNewAlleleNDNI 2459  1.08%
OkRefSameInsRev  2010  0.89%
OkRefAltSwitchIndelRpts     1738  0.77%
OkRefSameNDNIRev 1575  0.69%
RefNewAlleleDelRefChanged  1429  0.63%
RefNewAlleleInsSameRef      1181  0.52%
OkRefAltSwitchWithGap 871   0.38%
RefNewAlleleDelSameRef      512   0.23%
OkRefAltSwitchIndelFlank    389   0.17%
RefNewAlleleIndelNoSwitch  380   0.17%
RefNewAlleleInsRefChanged  313   0.14%
OkRefAltSwitchDelToIns      228   0.10%
INFO/AF    167   0.07%
```

```
CrossMap
data=indel primary=38 luft=t2t tool=CrossMap
Lifted OkRefSameIndel: 155695
Lifted OkRefSameNotLeftAnc: 30654
Lifted RefSplitInChain: 2554
Failed RefSplitInChain: 9593
Lifted RefNotMappedInChain: 2127
Failed RefNotMappedInChain: 3561
Lifted OkRefSameDelRev: 3823
Lifted RefMultiAltSwitchIndel: 3002
Lifted RefNewAlleleNotLeftAnc: 2835
Lifted RefNewAlleleNDNI: 2459
Lifted OkRefSameInsRev: 2010
Lifted OkRefAltSwitchIndelRpts: 1738
Lifted OkRefSameNDNIRev: 1575
Lifted RefNewAlleleDelRefChanged: 1429
Lifted RefNewAlleleInsSameRef: 1181
Failed OkRefAltSwitchWithGap: 871
Lifted RefNewAlleleDelSameRef: 512
Lifted OkRefAltSwitchIndelFlank: 389
Lifted RefNewAlleleIndelNoSwitch: 380
Lifted RefNewAlleleInsRefChanged: 313
Lifted OkRefAltSwitchDelToIns: 228
Lifted INFO/AF: 167

LiftoverVcf
data=indel primary=38 luft=t2t tool=gatk
Lifted OkRefSameIndel: 155695
Lifted OkRefSameNotLeftAnc: 30123
Failed OkRefSameNotLeftAnc: 531
Failed RefSplitInChain: 12147
Failed RefNotMappedInChain: 5688
Lifted OkRefSameDelRev: 3535
Failed OkRefSameDelRev: 288
Lifted RefMultiAltSwitchIndel: 1695
Failed RefMultiAltSwitchIndel: 1307
Lifted RefNewAlleleNotLeftAnc: 53
Failed RefNewAlleleNotLeftAnc: 2782
Lifted RefNewAlleleNDNI: 44
Failed RefNewAlleleNDNI: 2415
Lifted OkRefSameInsRev: 1998
Failed OkRefSameInsRev: 12
Lifted OkRefAltSwitchIndelRpts: 1735
Failed OkRefAltSwitchIndelRpts: 3
Lifted OkRefSameNDNIRev: 1271
Failed OkRefSameNDNIRev: 304
Lifted RefNewAlleleDelRefChanged: 66
Failed RefNewAlleleDelRefChanged: 1363
```

```
Lifted RefNewAlleleInsSameRef: 1181
Failed OkRefAltSwitchWithGap: 871
Lifted RefNewAlleleDelSameRef: 475
Failed RefNewAlleleDelSameRef: 37
Lifted OkRefAltSwitchIndelFlank: 389
Lifted RefNewAlleleIndelNoSwitch: 380
Lifted RefNewAlleleInsRefChanged: 121
Failed RefNewAlleleInsRefChanged: 192
Lifted OkRefAltSwitchDelToIns: 1
Failed OkRefAltSwitchDelToIns: 227
Failed INFO/AF: 167
```

5) SNPs – GRCh38 to T2T

```
Genozip
Showing counts of o$TATUS (did_i=17). Total items=4082903 Number of
categories=7
OkRefSameSNP	2288044	56.04%
OkRefAltSwitchSNP	1705586	41.77%
RefNotMappedInChain	61771	1.51%
OkRefSameSNPRev	22266	0.55%
RefNewAlleleSNP	1934	0.05%
OkNewRefSNP	1869	0.05%
RefMultiAltSwitchSNP	1433	0.04%

CrossMap
data=snp primary=38 luft=t2t tool=CrossMap
Lifted OkRefSameSNP: 2288044
Failed OkRefAltSwitchSNP: 1705586
Failed RefNotMappedInChain: 61771
	Lifted OkRefSameSNPRev: 22266
Lifted RefNewAlleleSNP: 1934
Lifted OkNewRefSNP: 1869
Lifted RefMultiAltSwitchSNP: 1433

LiftoverVcf
data=snp primary=38 luft=t2t tool=gatk
Lifted OkRefSameSNP: 2288044
Lifted OkRefAltSwitchSNP: 1705586
Failed RefNotMappedInChain: 61771
Lifted OkRefSameSNPRev: 22266
Failed RefNewAlleleSNP: 1934
Failed OkNewRefSNP: 1869
Failed RefMultiAltSwitchSNP: 1433
```

6) ClinVar – GRCh38 to T2T

```
Genozip
Showing counts of o$TATUS (did_i=17). Total items=969486 Number of
categories=22
OkRefSameSNP    857612    88.46%
OkRefSameIndel  86095 8.88%
OkRefAltSwitchSNP    11055 1.14%
OkRefSameNotLeftAnc   5972  0.62%
OkRefSameSNPRev  2058  0.21%
RefNewAlleleInsSameRef     1071  0.11%
RefSplitInChain  1003  0.10%
RefNotMappedInChain   859   0.09%
OkRefAltSwitchIndelRpts    734   0.08%
RefNewAlleleSNP  732   0.08%
RefNewAlleleDelRefChanged  520   0.05%
RefNewAlleleNotLeftAnc     457   0.05%
OkRefAltSwitchWithGap 432   0.04%
RefNewAlleleDelSameRef     340   0.04%
OkRefAltSwitchIndelFlank   152   0.02%
RefNewAlleleInsRefChanged  111   0.01%
OkRefSameDelRev  89    0.01%
OkRefAltSwitchDelToIns     53    0.01%
OkRefAltSwitchNotLeftAnc   50    0.01%
OkRefSameInsRev  48    0.00%
RefNewAlleleIndelNoSwitch  42    0.00%
ChromNotInPrimReference    1     0.00%

CrossMap
data=clinvar primary=38 luft=t2t tool=CrossMap
Lifted OkRefSameSNP: 857612
Lifted OkRefSameIndel: 86095
Failed OkRefAltSwitchSNP: 11055
Lifted OkRefSameNotLeftAnc: 5972
Lifted OkRefSameSNPRev: 2058
Lifted RefNewAlleleInsSameRef: 1071
Lifted RefSplitInChain: 106
Failed RefSplitInChain: 897
Lifted RefNotMappedInChain: 68
Failed RefNotMappedInChain: 791
Lifted OkRefAltSwitchIndelRpts: 734
Lifted RefNewAlleleSNP: 732
Lifted RefNewAlleleDelRefChanged: 520
Lifted RefNewAlleleNotLeftAnc: 456
Failed RefNewAlleleNotLeftAnc: 1
Lifted OkRefAltSwitchWithGap: 1
Failed OkRefAltSwitchWithGap: 431
Lifted RefNewAlleleDelSameRef: 340
Lifted OkRefAltSwitchIndelFlank: 152
Lifted RefNewAlleleInsRefChanged: 111
Lifted OkRefSameDelRev: 89
Lifted OkRefAltSwitchDelToIns: 53
Lifted OkRefAltSwitchNotLeftAnc: 16
Failed OkRefAltSwitchNotLeftAnc: 34
Lifted OkRefSameInsRev: 48
Lifted RefNewAlleleIndelNoSwitch: 42
Failed ChromNotInPrimReference: 1

LiftoverVcf
data=clinvar primary=38 luft=t2t tool=gatk
Lifted OkRefSameSNP: 857612
Lifted OkRefSameIndel: 86095
Lifted OkRefAltSwitchSNP: 11055
Lifted OkRefSameNotLeftAnc: 5972
Lifted OkRefSameSNPRev: 2058
Lifted RefNewAlleleInsSameRef: 1071
Failed RefSplitInChain: 1003
Failed RefNotMappedInChain: 859
Lifted OkRefAltSwitchIndelRpts: 734
Failed RefNewAlleleSNP: 732
Failed RefNewAlleleDelRefChanged: 520
Failed RefNewAlleleNotLeftAnc: 457
Failed OkRefAltSwitchWithGap: 432
```

```
Lifted RefNewAlleleDelSameRef: 340
Lifted OkRefAltSwitchIndelFlank: 152
Lifted RefNewAlleleInsRefChanged: 2
Failed RefNewAlleleInsRefChanged: 109
Lifted OkRefSameDelRev: 87
Failed OkRefSameDelRev: 2
Failed OkRefAltSwitchDelToIns: 53
Failed OkRefAltSwitchNotLeftAnc: 50
Lifted OkRefSameInsRev: 48
Lifted RefNewAlleleIndelNoSwitch: 42
Failed ChromNotInPrimReference: 1
```

Thesis discussion

Thesis summary and significance

In this PhD project, my aim was to advance the field of genomic data compression, both at the theoretical level by devising new algorithms and methods to achieve better compression, as well as at the applied level by assembling the best software engineering models to create a practical, reliable tool that researchers and clinicians can benefit from in the years and decades to come. My hope is that with better storage management of genomic data, the generation of genomic data and its usage will become more economically feasible, and this small contribution of mine will have some impact at accelerating the pace of the genomics revolution, in particular as it pertains to the clinical space, which translates directly to saving or improving human lives.

Chapters 1 and 2

In chapters 1 and 2, I described the invention of several new methods for compressing various elements of genomic data, as well as the architectural framework of the Genozip software. Using a series of benchmarks, I demonstrated how the new compression methods implemented in Genozip are superior to previous approaches in many important cases.

It is my opinion that the file formats that Genozip handles will still be the file formats used decades ahead, despite their shortcomings. Indeed, there is current work being conducted to address the limitations of genomic file formats - most notably, to support expression of genomic coordinates on a pangenome graph rather than a linear reference [1] to better address worldwide human genetic diversity and capture their underlying ancestries. A file format called GAM [2] is proposed as a replacement for BAM that is suitable for expressing read alignment to a pangenome graph, and the VG format [3] is proposed as a pangenome graph replacement for both a linear reference genome, which is typically expressed in FASTA file, and the description of genetic variants within specific samples, for which the VCF format is the current de-facto standard.

It is my estimation that these new formats will, with all likelihood, find their place in areas of research where their advantages are critical, such as research of structural variants across populations. However, I do not anticipate that they will be adopted in the broad research and clinical community, as the current formats are entrenched with hundreds of tools that are dependent upon them, petabytes of legacy data, and the scores of trained users churned out by biological, medical, and bioinformatics programs worldwide over the past 15 years. More crucially, for most common use cases, there is no obvious compelling reason to switch yet.

This conviction is based on following analogous development of standards in a related rapidly evolving industry - the Internet. During my time as a computer science undergraduate student in the early 90s, there was work going on to solve the limitations of IPv4, the Internet Protocol first used in the early 1980s that was designed to support up to 4 billion devices - seemingly more than enough, even in the minds of the wildest imaginators of the time. In 1995, the Internet Engineering Task Force (IETF) - the body that governs the technical standards of the Internet - published what became eventually known as IPv6 as a recommendation [4] and as a standard in 1998 [5]. The industry consensus was that IPv6 would rapidly replace IPv4, as the latter is ill-suited to serve as the backbone of a network that connects billions of devices worldwide. However, the end-users of computers and phones and their Internet Service Providers had no compelling reason to switch, and the limitations of IPv4 were addressed with a patchwork of hacky enhancements. As a result, to this day, 25 years or so after the advent of IPv6 that is without question superior to IPv4, the world is predominantly still using IPv4. Similarly, without a compelling reason to switch for a broad population of bioinformatics users, I anticipate that we will still be using FASTA, FASTQ, BAM/CRAM and VCF formats in the decades to come, and Genozip will still be around to make their usage more efficient.

Chapter 3

In Chapter 3 I tackled the file format - VCF - that is often used as the workhorse of analysis in a wide range of genomics subfields, including clinical genetics, population genetics and others. This file format is over a decade old now and comes with many limitations, which include not utilising common data standards such as XML or JSON, high levels of data redundancy (and hence the need for compression), and is sufficiently complex to make error-checking difficult [6]. However, as discussed above, it is my forecast that the VCF format will continue to be the primary format for expressing genetic variation for many years to come, despite its known shortcomings, and will be improved incrementally with point enhancements. In Chapter 3 and the dissertation Appendix, I proposed one such enhancement - a new extension of VCF, called DVCF, and implemented it in Genozip. DVCF allows users and bioinformatics tools to access genetic variants using coordinates of two different reference genomes concurrently. While Genozip is designed to be a compression tool, not an analysis tool, a by-product of its architecture is a fine grained internal representation of genomic data, which then allows development of innovative transformations of the data. DVCF is one such example, and others might be added in the future.

The importance of DVCF is elevated due to human genomics research still transitioning from GRCh37 to GRCh38 as the standard version of the human reference genome, and as a result, it is not uncommon for a dataset to be processed in a bioinformatics pipeline that contains steps alternating between these two reference genomes. GRCh38 was first introduced in 2013, almost a decade ago, and the reason the transition to it is incomplete is a similar lack of compelling reason to switch as described earlier. Compounding this problem is the recent availability of a new human reference genome, T2T-CHM13, provided by the Telomere-to-Telomere consortium [7]. Gradual adoption of this reference genome by some will inevitably result in three human reference genomes being commonly used by the research community.

Conclusion and future directions

Reflecting on the hypothesis "*methods tailored to the structure of genomic data will improve compression rates*": Genozip has been implemented based on this hypothesis - and as demonstrated in chapters 1, 2, this approach indeed succeeded in improving compression rates in a wide range of cases.

There are two equally important axes that are both required to sustain and improve Genozip in the years to come. The first, is on the algorithmic axis - continuous improvement of the algorithms, and support for new sequencing technologies and file formats created by new bioinformatics methods as they emerge. The second, which is often overlooked in the academic world, is creating a financial model that will sustain ongoing development and support of Genozip.

On the algorithmic side, the current version of Genozip offers the best compression of genomic files available for a wide range of common genetic data cases. However, there is still work to be done to improve the compression, as there are many more data redundancies to be exploited, as well as better compression for a variety of genomic file types that are the products of common tools used in the field.

In addition, there is a need to make Genozip easier to integrate into existing bioinformatics pipelines that requires only minimal modifications to the pipeline. There are a number of different approaches for achieving this: one would be including access to Genozip into htslib, the underlying library used by many software packages for reading genomic files, while another would be presenting compressed files as a virtual filesystem.

New sequencing technologies, which increasingly rely on complex machine learning algorithms for base calling (e.g., Oxford Nanopore Technology, Ultima Genomics), pose new compression challenges as the statistical properties of their data differ significantly from traditional high-throughput DNA sequencing data, therefore different methods are needed for their compression. A substantial research effort will be required for solving this problem.

An interesting potential application of DVCF not yet explored, is using it to generate VCF files describing variants in homologous genes in related species. This remains an area of future research. It might also be useful to extend DVCF from being “dual coordinate” to supporting an arbitrary number of reference files.

Also for DVCF, the current lift-over algorithm covers significantly more cases than previous lift-over software packages, but it still doesn’t cover all cases. Most notably, it doesn’t handle structural variants (such variants remain in the DVCF file as “single-coordinate variants”, encoded in the VCF header). The reason for this is that it is extremely tricky to lift over such variants against a chain file that itself might contain structural changes. This is an area for future research.

As good as the DVCF format may be, it will have little impact if it is not widely adopted by the research community. To this end, it is my intention to engage with the VCF specification community to attempt to get DVCF officially supported.

The sequence of development of the ideas that lead to DVCF, resulted in it being implemented in Genozip, which is primarily used as a compression tool. It might be useful to also have a standalone liftover tool based on the DVCF format which is separate from Genozip and does not store data in the Genozip compressed format.

Genozip, at its core, is a system for storing genomic data in an efficient way. Decompressing is simply re-writing the data from Genozip format back to its original format. However, there could be other useful ways to manipulate the data, or present it in new and useful formats, as demonstrated in DVCF. This is also an area for future research.

As data security is a growing concern in the area of genomics, Genozip comes with a built-in encryption capability. The field of encryption, and in particular encryption algorithms resistant to quantum-computing attacks, is evolving rapidly. For this reason, adding encryption algorithms is supported by the current design of the Genozip file format - it would be fairly straightforward to add additional encryption algorithms in the future. While quantum computing is still only on the distant horizon of the average bioinformatician, there is some hope that AES-256, the encryption algorithm currently used by Genozip, being S-Box-based rather than based on a mathematical problem as in the RSA or Elliptic Curves algorithms, is inherently more resistant to quantum attack algorithms (Rao et al., 2017) following the concept introduced by (Shor, 1994). Encryption algorithms recently selected by NIST for their quantum-resistance might offer even more protection (Boutin, 2022).

Apache ORC is a system for storing columnar data which is gaining popularity. To the extent that in the future, usage of this system becomes common in the bioinformatics space, it would be a good idea to extend Genozip to seamlessly integrate with it.

On the financial side, it is now clear that the open source model has significant challenges when it comes to bioinformatics tools. Hundreds of potentially useful tools, which cumulatively consumed a huge amount of effort and resources to build, are left to "decompose" in github - i.e., becoming increasingly useless as needs and data formats evolve, and as their original developers move on. The idea that "anyone" can maintain these projects, while attractive, rarely materialises - most projects have no other contributors beyond their original developers, and other developers would rarely be interested in maintaining an existing project with neither payment nor potential publication being viable. The successful open source tools that are continuously evolving are typically ones that are supported by engineers who receive a regular salary to do so, from organisations who are either taxpayer funded or who have a commercial interest to keep a particular open source project alive.

It is therefore clear to me that in order for Genozip to continue to evolve into the future, there is a need to commercialise it. The model I have chosen is to continue and provide it for free for academic research purposes, while charging users who use Genozip as part of their business - primarily in the clinical space, in the bioinformatics-services space and in the product-development space (biotech, agrotech etc). In addition, I have made the choice to keep the source code available on github (albeit with a restrictive licence, rather than open source one) to ensure that files compressed with Genozip today would be accessible in the decades to come regardless of my own personal circumstances, as well as to encourage other compression researchers to critically review Genozip.

All said, it is with great satisfaction that I inspect the list of research labs that are using Genozip (Appendix 3), a list that is growing daily. My hope is that Genozip will prove to be a useful tool for researchers and clinicians around the world, and have a small contribution to the advancement of using genetics for improving human lives - that shall be my true reward for this effort.

References

1. Eizenga JM, Novak AM, Sibbesen JA, Heumos S, Ghaffaari A, Hickey G, et al. Pangenome Graphs. Annu Rev Genomics Hum Genet. 2020;21: 139–162. doi:10.1146/annurev-genom-120219-080406

2. vg Wiki. Github; Available: https://github.com/vgteam/vg

3. Hickey G, Heller D, Monlong J, Sibbesen JA, Sirén J, Eizenga J, et al. Genotyping structural variants in pangenome graphs using the vg toolkit. Genome Biol. 2020;21: 35. doi:10.1186/s13059-020-1941-7

4. RFC 1752 - the Recommendation for the IP Next Generation Protocol. [cited 13 Apr 2022]. Available: https://datatracker.ietf.org/doc/html/rfc1752

5. Deering S, Hinden R. Rfc 2460-internet protocol, version 6 (ipv6) specification, 1998. One citation on. 2014; 19. Available: https://datatracker.ietf.org/doc/html/rfc2460

6. Garrison E, Kronenberg ZN, Dawson ET, Pedersen BS, Prins P. Vcflib and tools for processing the VCF variant call format. bioRxiv. 2021. p. 2021.05.21.445151. doi:10.1101/2021.05.21.445151

7. Nurk S, Koren S, Rhie A, Rautiainen M, Bzikadze AV, Mikheenko A, et al. The complete sequence of a human genome. Science. 2022;376: 44–53. doi:10.1126/science.abj6987

8. Boutin,C. (2022) NIST announces first four quantum-resistant cryptographic algorithms | NIST.

9. Rao *et al.* The AES-256 cryptosystem resists quantum attacks. *Int. J. Advanced Research in Computer Science*, 8(3): 404-408

10. Shor,P.W. (1994) Algorithms for quantum computation: discrete logarithms and factoring. In, *Proceedings 35th Annual Symposium on Foundations of Computer Science.*, pp. 124–134.

Appendix 1: The DVCF Specification

The Variant Call Format Dual Coordinate Extension (DVCF) Specification

Version 1.2
August 8, 2021

Written by: Divon Lan, divon@genozip.com

Citing (pre-print):
Lan, D (2021) The Variant Call Format - Dual Coordinate Extension (DVCF) Specification (preprint)
doi:10.6084/m9.figshare.14685816

Statement of Authorship

Title of Paper	The Variant Call Format Dual Coordinate Extension (DVCF) Specification
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	Published as a pre-print to avoid assigning IP rights to a journal, as the intention is to use this paper as a basis for a proposed industry standard.

Principal Author

Name of Principal Author (Candidate)	Divon Mordechai Lan		
Contribution to the Paper	Wrote the paper		
Overall percentage (%)	100%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature	[redacted]	Date	21/04/2022

1. Background

This specification is fully compatible with the VCFv4.3 specification, and extends it. It is also fully compatible with VCF v4.1 and v4.2.

The specification defines a derived format of VCF, fully compliant with the VCF specification, which is called the Dual Coordinate VCF file (or *DVCF*). A DVCF file contains information about genetic variants in two different coordinate systems. The key feature of DVCF is that it can be *rendered* in two different ways - the *Primary rendition* and *Luft rendition*. Both these renditions are VCF specification-compliant files, that contain precisely the same information, merely *rendered* in two different coordinate systems.

Since these two renditions contain precisely the same information, they can be losslessly *cross-rendered* back and forth. Cross-rendering is a fast operation that does not require a reference or chain file.

Once a VCF file is *lifted* to a Dual Coordinate VCF file - it can be processed through an analytical pipeline, and since the data can be rendered in either coordinate system, each stage of the pipeline can arbitrarily operate on either coordinate system. Importantly, the rendering continues to work as fields and annotations are added, removed or modified, as the data works its way down the pipeline.

This specification was intentionally made to be similar to the VCF specification in format, structure and terminology, and is designed to be read alongside it. All the definitions and requirements that appear in the VCF specification apply here as well, and they are not repeated in this document.

A reference implementation is provided in Genozip, available on genozip.com (Lan *et al.*, 2021, 2020).

This specification was written in the context of a PhD project at the University of Adelaide, Australia. I wish to thank my PhD supervisors Assoc. Prof. Bastien Llamas, Dr. Yassine Souilmi and Dr. Ray Tobler, as well as my wife, Channé Suy Lan, for their support which has been absolutely essential.

2. Definitions

- A *Source VCF* is any VCF file that is compliant with the VCF specification.

- The *Primary coordinate system* is the coordinate system of the Source VCF.

- The *Luft coordinate system* is the other coordinate system in which the variant data will be expressed ("Luft" being a made-up past participle of "Lift").

- A *Primary rendition* and a *Luft rendition* are VCF files expressed in the Primary and Luft coordinates respectively, which are equivalent to each other and contain all the information of a Source VCF along with all the information needed to *cross-render* them (see below). A DVCF is always rendered in one or both of these two renditions, and this specification defines no other representation of a DVCF other than the renditions.

- A *Lifter* is a software functionality that converts, or *Lifts*, a Source VCF to a DVCF. It may use auxiliary information such as a reference file in the Luft coordinates and a chain file.

- A *Renderer* is a software functionality that generates the Primary and Luft renditions. It may *cross-render* a Primary rendition to a Luft one or vice versa, or may generate a Primary or Luft rendition from some other data. Cross-rendering does not require any external information beyond the input DVCF file itself. Specifically, it does not require a reference file or a chain file.

- A *Dual Coordinates VCF implementation* (or just *implementation* for brevity) means a particular software package including the functionalities of a Lifter and/or a Renderer.

3. Scope of this specification

This specification defines the formats of the DVCF Primary and Luft renditions.

It does not define the algorithms of a Lifter or Renderer, however it does set constraints on them, to ensure that the Primary and Luft renditions contain precisely the same information, and to ensure interoperability between implementations. While adhering to these constraints, different implementations of *Lifters* and *Renderers* might operate differently to address different needs.

Complying with this specification will ensure that the resulting files are interoperable across different systems.

It is desirable that any software that converts a VCF file from one coordinate system to another, shall be capable of outputting VCF files in DVCF format.

4. An Example

The following are the two *renditions* of the same DVCF - they are two files containing precisely the same information - the first file is the *Primary rendition* in GRCh37 coordinates, and the second is the *Luft rendition* in GRCh38 coordinates. This DVCF contains 3 variants and 2 samples.

A *Primary rendition VCF* file:

```
##fileformat=VCFv4.2
##dual_coordinates=PRIMARY
##chain=file:///data/GRCh37_to_GRCh38.chain.genozip
##reference=file:///references/grch37/reference.bin
##luft_reference=file:///data/GRCh38_full_analysis_set_plus_decoy_hla.ref.genozip
##FILTER=<ID=PASS,Description="All filters passed">
##FORMAT=<ID=AD,Number=R,Type=Integer,Description="Allelic depths for the ref and alt alleles",RendAlg=”R">
##FORMAT=<ID=AF,Number=A,Type=Float,Description="Allele fractions for alt alleles in the order listed",RendAlg="A_1">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype",RendAlg="GT">
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="Normalized, Phred-scaled likelihoods for genotypes",RendAlg="G">
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes, for each ALT allele",RendAlg="A_AN">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes",RendAlg="NONE">
##INFO=<ID=LUFT,Number=4,Type=String,Description="Info for rendering variant in LUFT coords",RendAlg="NONE">
##INFO=<ID=PRIM,Number=4,Type=String,Description="Info for rendering variant in PRIMARY coords",RendAlg="NONE">
##INFO=<ID=Lrej,Number=1,Type=String,Description="Reason variant was rejected for LUFT coords",RendAlg="NONE">
##INFO=<ID=Prej,Number=1,Type=String,Description="Reason variant was rejected for PRIMARY coords",RendAlg="NONE">
##contig=<ID=1,length=249250621>
##luft_contig=<ID=chr1,length=248956422>
#CHROM POS    ID     REF    ALT    QUAL   FILTER INFO   FORMAT Person1       Person2
1      10285  .      T      C      4.4    PASS   AC=3;AN=4;LUFT=chr1,10285,T,-    GT:AD:AF:PL  0/1:31,18:0.367:37,0,46    1/1
1      329162 .      A      T      4.6    PASS   AC=3;AN=4;LUFT=chr1,248466248,T,-GT:AD:AF:PL  0/1:28,9:0.3:36,0,0 1/1
1      366043 .      CA     A      100    PASS   Lrej=RefTooLong     GT     1|0    0|0
```

A *Luft rendition VCF* file corresponding to the *Primary rendition* on the previous page:

```
##fileformat=VCFv4.2
##dual_coordinates=LUFT
##chain=file:///data/GRCh37_to_GRCh38.chain.genozip
##reference=file:///data/GRCh38_full_analysis_set_plus_decoy_hla.ref.genozip
##primary_reference=file:///references/grch37/reference.bin
##FILTER=<ID=PASS,Description="All filters passed">
##FORMAT=<ID=AD,Number=R,Type=Integer,Description="Allelic depths for the ref and alt alleles",RendAlg="R">
##FORMAT=<ID=AF,Number=A,Type=Float,Description="Allele fractions for alt alleles in the order listed",RendAlg="A_1">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype",RendAlg="GT">
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="Normalized, Phred-scaled likelihoods for genotypes",RendAlg="G">
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes, for each ALT allele",RendAlg="A_AN">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes",RendAlg="NONE">
##INFO=<ID=LUFT,Number=4,Type=String,Description="Info for rendering variant in LUFT coords",RendAlg="NONE">
##INFO=<ID=PRIM,Number=4,Type=String,Description="Info for rendering variant in PRIMARY coords",RendAlg="NONE">
##INFO=<ID=Lrej,Number=1,Type=String,Description="Reason variant was rejected for LUFT coords",RendAlg="NONE">
##INFO=<ID=Prej,Number=1,Type=String,Description="Reason variant was rejected for PRIMARY coords",RendAlg="NONE">
##primary_contig=<ID=1,length=249250621>
##contig=<ID=chr1,length=248956422>
##primary_only=1    366043 .      CA     A      100    PASS   Lrej=RefTooLong     GT     1|0    0|0
#CHROM POS    ID     REF    ALT    QUAL   FILTER INFO   FORMAT Person1      Person2
chr1   10285  .      T      C      4.4    PASS   AC=3;AN=4;PRIM=1,10285,T,- GT:AD:AF:PL  0/1:31,18:0.367:37,0,46    1/1
chr1   248466248     .      T      A      4.6    PASS   AC=1;AN=4;PRIM=1,329162,A,-      GT:AD:AF:PL  1/0:9,28:0.7:0,0,36 0/0
```

5. Meta-information lines

The following meta-information lines are added or modified. They may appear in any order.

5.1. Coordinates

```
##dual_coordinates=PRIMARY
```

This field is required.

Permitted values: `PRIMARY`, `LUFT`. Defines the coordinates of the current rendition.

5.2. Chain file URL

This field is recommended.

```
##chain=file:///data/GRCh37_to_GRCh38.chain.genozip
```

The URL of the chain file used by the Lifter to generate this DVCF. The file format and naming conventions of the chain file are implementation-specific and out of scope of this specification.

5.3. Reference files' URLs

These fields are recommended.

In a Primary rendition it is recommend to include the `##reference` and `##luft_reference` lines. The former contains the URL of the reference file of the Primary coordinates, and `##luft_reference` contains the URL of the reference file of the Luft coordinates.

```
##reference=file:///data/hg19.p13.plusMT.full_analysis_set.ref.genozip
```

```
##luft_reference=file:///data/GRCh38_full_analysis_set_plus_decoy_hla.r
ef.genozip
```

Similarly, in a Luft rendition, it is recommended to include `##reference` (Luft coordinates reference file) and `##primary_reference`:

```
##reference=file:///data/GRCh38_full_analysis_set_plus_decoy_hla.ref.ge
nozip

##primary_reference=file:///data/hg19.p13.plusMT.full_analysis_set.ref.
genozip
```

The file format and naming conventions of the reference files are implementation-specific and out of scope of this specification.

5.4. RendAlg attribute of ##INFO and ##FORMAT

```
##FORMAT=<ID=GL,Number=G,Type=Float,Description="Genotype
Likelihoods",RendAlg="G">

##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in
genotypes, for each ALT allele, in the same order as
listed",RendAlg="A_AN">
```

The `RendAlg` attribute *must* be present in all `##INFO` and `##FORMAT` meta-information lines.

The *Renderer must* add `RendAlg` to `##INFO` or `##FORMAT` meta-information lines that are missing them. It *must not* modify a RendAlg value if one is already present. Lines might be missing `RendAlg` if, for example, the file acquired additional `INFO` or `FORMAT` fields in an analysis step.

5.5. LUFT, PRIM, Lrej and Prej

DVCF files must contain the following four meta-information lines defining `INFO/LUFT`, `INFO/PRIM`, `INFO/Lrej` and `INFO/Prej`. The `ID`, `Number`, `Type` and `RendAlg` attributes *must* appear as below, other attributes (such as `Description`) are optional.

```
##INFO=<ID=LUFT,Number=4,Type=String,RendAlg="NONE">
##INFO=<ID=PRIM,Number=4,Type=String,RendAlg="NONE">
##INFO=<ID=Lrej,Number=1,Type=String,RendAlg="NONE">
##INFO=<ID=Prej,Number=1,Type=String,RendAlg="NONE">
```

5.6. Contigs

The `##contig` key is as defined in the VCF specification. It refers to contigs of the current coordinates.

In a Primary rendition file, meta-information lines with a `##luft_contig` key may exist, and have the same format as `##contig`. They describe the contigs that appear in the Luft rendition. Similarly, a Luft rendition files may contain `##primary_contig` keys, describing the contigs in the Primary rendition. It is recommended that a DVCF file includes a `##luft_contig` or `##primary_contig` line for each contig that appears in the file.

Note that there is no requirement for a 1:1 mapping between contigs - indeed, it is possible that two variants with a particular contig in one coordinate system, are mapped to two different contigs on the other coordinate system.

5.7. ##primary_only and ##lift_only

In the *Primary rendition* `##luft_only` meta-information lines contain variants that are not renderable in Primary coordinates, and similarly, in the *Luft rendition*, `##primary_only` meta-information lines contain variants that are not renderable in Luft coordinates. Following the key at the '=' character, the remainder of the line is a normal VCF data line as defined in the VCF specification.

6. Variants

6.1. Overview

Each variant in the DVCF can be a dual-coordinate variant, or it could be a primary-coordinates-only variant or a luft-coordinates-only variant. The latter two cases happen when a variant can only be expressed in one of the coordinates but not in the other, in which case we also refer to it as *rejected* from the other coordinates. There are many reasons a variant can be rejected, discussed below.

Each variant, in both renditions, contains exactly one *DVCF tag*, which is an INFO field carrying DVCF-related information - one of: PRIM, LUFT, Prej or Lrej.

A dual-coordinate variant appears as a normal VCF variant in both renditions, and contains an INFO/LUFT field in the Primary rendition with the information needed to cross-render this variant to Luft coordinates, and similarly, in the Luft rendition, it contains an INFO/PRIM field with the information needed to cross-render the variant to Primary coordinates.

A primary-coordinates-only variant contains an INFO/Lrej field with the reason it was rejected for rendering in Luft coordinates. In the Primary rendition, the variant appears as a normal VCF data line, while the Luft rendition, this variant will appear as-is (i.e. in Primary coordinates) in a meta-information line with the key `##primary_only`.

Likewise, luft-coordinates-only variants have a INFO/Prej field, and appear as a data line in the Luft rendition, and as a meta-information line they key `##luft_only` in the Primary rendition.

6.2. CHROM, POS, REF, ALT

The *Lifter*, given a Source VCF and in consultation with external information, typically a reference file in the Luft coordinates and a chain file, should calculate the CHROM, POS, REF fields in the Luft coordinates. These *must* be biologically correct (the exact definition of *biologically correct* is left to the implementation) and either generate a *dual-coordinate variant* or a *primary-only variant*, i.e. one that was rejected from the Luft coordinates.

A *dual-coordinate variant* appears as a VCF data line in both Primary and Luft renditions:

- A dual-coordinate variant in the Primary rendition has the CHROM, POS, REF and ALT fields appear as in the Source VCF.

  It also has a INFO/LUFT field that contains four values, for example:

”`LUFT=chr2,1000000,G,X`”. The first two values are the CHROM and POS of this variant in Luft coordinates. The third is the Luft reference value of this variant. The fourth value, which we call XSTRAND, must be one of two options: it is `X` (capital letter X) if the alignment in the chain file which includes this locus has opposite strands for the Primary and Luft references and `-` (hyphen) if the strands are the same.

- A dual-coordinate variant in the Luft rendition has the values of CHROM, POS and REF as they appear in INFO/LUFT in the Primary rendition, and has the ALT calculated as described below.

  It also has an INFO/PRIM field that contains four values, of the same structure as INFO/LUFT: the first three values are the CHROM, POS and REF in Primary coordinates, and the fourth is the XSTRAND of this variant. XSTRAND *must* be the same value as in INFO/LUFT.

A primary-only variant appears in both renditions in primary coordinates. In the Primary rendition, it appears as a normal VCF data line, while in the Luft rendition, it appears as a `##primary_only` meta-information line. Apart from the “`##primary_only=`” prefix, the meta-information line is identical to the VCF data line as it appears in the Primary rendition.

In both renditions a primary-only variant has an INFO/Lrej field which contains the reason for its rejection. This specification defines a number of standard reasons, and an implementation may add additional reasons. The standard reasons are listed in section 7 of this document.

The *lifter* uses the information in CHROM, POS, REF and ALT as well as external information such as a chain file and a Luft reference file, to generate either a INFO/LUFT of INFO/Lrej field, while the *renderer* uses the information in the CHROM, POS, REF, ALT and INFO/LUFT or INFO/PRIM fields to calculate the CHROM, POS, REF, ALT and INFO/PRIM or INFO/LUFT respectively, of the other rendition, or it may reject the cross-rendering resulting in a Primary-only variant with an INFO/Lrej field, or a Luft-only variant with INFO/Prej field.

If the REF changes between Primary and Luft references, a *lifter* is free to either lift the variant or reject it, with the rejection reason placed in INFO/Lrej being one of the standard reasons listed in section 7, or an implementation-defined reason. If as a result of the REF change, the number of alleles grows because Luft REF is not any of the Primary alleles, then the Primary REF *must* be last on the Luft ALT list.

When calculating the ALT field, the algorithm used by the *lifter* and *renderer must:*

1. Be biologically-correct (the definition of *biologically correct* is left to the implementation).

2. Be precisely invertible, so that cross-rendering from the Luft rendition to the Primary rendition and back to the Luft rendition, as well as Primary ➝ Luft ➝ Primary results in the precisely preserving the REF and ALT fields, including the case (upper or lower) of each character.

A *renderer,* when cross-rendering a file, may encounter variants that are lacking a DVCF tag. This may happen, for example, when a non-DVCF VCF file is merged into a DVCF file, resulting in variants added that are lacking a DVCF tag. In this case, these variants become single-coordinate variants (in the coordinates of the current rendition), and the *renderer must* set the DVCF tag to `Lrej=AddedVariant` (Primary-only variant) or `Prej=AddedVariant` (Luft-only variant).

A *renderer*, when cross-rendering a file, may encounter variants that have both a PRIM/LUFT field as well as a Prej/Lrej one. This can happen when rendering a DVCF that is a result of merging two DVCF files. The renderer *must* discard one of these fields.

If the *renderer*, when cross-rendering a Luft variant, rejects it - that variant becomes a Luft-only variant, the DVCF tag is set to Prej, and it appears in the Primary rendition as a `##luft_only` meta-information line, similar to the `##primary_only` meta-information line described above.

6.3. INFO and FORMAT fields - RendAlg

Each FORMAT and INFO tag has a RendAlg algorithm associated with it. If the tag has no `##INFO` or `##FORMAT` meta-information line, or the line is lacking a RendAlg attribute, the implementation may decide to apply any of the RendAlgs. For example, it may decide that the field INFO/AF, in case it has no `##INFO` meta-information line, will use the A_1 RendAlg.

Each RendAlg has an *ID*, which appears in the `RendAlg` attribute of the `##INFO` and `##FORMAT` meta-information lines, a *Trigger*, which is a description of the circumstances in which the RendAlg should be applied, and an *Action*, which is a description of the transformation of the data that occurs when the *Trigger* is activated.

The table below lists the standard RendAlgs. An implementation may or may not support any of the standard RendAlgs, and may also add additional RendAlgs. For the *REF change* trigger, it may support all or only certain types of REF changes. However, if a trigger which is supported by the implementation occurs for any particular variant, then each field of the variant that is assigned a standard RendAlg *must* be transformed according to the standard action listed.

When cross-rendering, a Renderer *must* cross-render every INFO and FORMAT field according to its *RendAlg*, if the *Trigger* has occurred.

If cross-rendering fails for a particular field, then the variant will have an INFO/Lrej or INFO/Prej field, with *Reason* set to the rejected INFO or FORMAT field name, for example `Lrej=INFO/END`.

Any RendAlg algorithm *must* be losslessly invertible. In other words, applying it to a variant in one rendition, and then applying the inverse algorithm to the resulting other rendition, must result in getting back the original rendition, precisely. For example, the GT RendAlg listed below, upon REF ⇆ALT switch of a bi-allelic, triploid variant, will flip allele numbers in an unphased FORMAT/GT field `0/1/1` to `1/0/0`. It may have been desirable to also sort the result as common in representation of unphased genotypes, so `1/0/0` becomes `0/0/1`. However, that would cause the loss of the information regarding the original order of allele values, and hence the non-existence of a losslessly invertible algorithm, and is therefore prohibited.

While in most cases the RendAlg will only modify the INFO or FORMAT field on which it triggered, it is not restricted in this way: A RendAlg algorithm may change, add or remove other fields of the variant, so long as it is losslessly invertible.

While a *Lifter* transforming a Source VCF to a DVCF in the Primary rendition need not cross-render INFO and FORMAT fields, it is recommended that it nevertheless validates that cross-rendering may be carried out successfully, and sets an INFO/Lrej field if not.

Note that *REF change* and *Strand reversal* are orthogonal events - just a strand reversal isn't a *REF change*, despite the REF being reverse complemented.

ID	**Triggered upon**	**Action**	**Recommended for**
NONE	Never	Do nothing	Fields that don’t require change
G	REF change	Re-order / expand the values of a field that has one value per genotype	Fields with Number=G, such as: FORMAT/GL
R	REF change	Re-order / expand the values of a field that has one value per allele	Fields with Number=R, such as: FORMAT/AD
R2	REF change	Re-order / expand the values of a field that has 2 values per allele	Fields with 2 values per allele, such as: FORMAT/SAC
A_1	REF change	Re-calculate / expand the values of a field, so that their sum plus the implied value for the REF allele is 1.	Fields with Number=A, whose values, including the implied value for REF, add up to 1. Examples: FORMAT/AF, INFO/AF
A_*tag*	REF change	Re-calculate / expand the values of a field, so that their sum plus the implied value for the REF allele equals the value in INFO/*tag*.	Fields with Number=A, whose values, including the implied value for REF, add up to the value in INFO/*tag*. Example: INFO/AC would have a RendAlg of A_AN.
MAX_*tag*|*tag...*	REF change	Value is the maximum of the INFO/*tag* values of the 1 or more tags listed.	INFO/MAX_AF
PLOIDY	REF change	Recalculate: value = (ploidy - value)	FORMAT/DS (bi-allelic)
GT	REF change	Re-assign / add allele numbers based on the new REF/ALT order. Alleles in the genotype are not reordered.	FORMAT/GT
XREV	Strand reversal	Reverse the order of the elements in the array	Fields with a value per base ACGT. E.g INFO/BaseCounts
ALLELE	Always	Value is identical to one of the alleles. It shall remain identical to that allele even if it changes order or is reverse-complemented and shifted	INFO/AA
END	Always	Recalculate the value so that (value - POS) remains unchanged	INFO/END

6.4. INFO and FORMAT fields - Tag Renaming

INFO and FORMAT meta-information lines may optionally have these attributes, possibly more than one of them, describing changes to their tag name when cross-rendering, conditional on a certain trigger occurring:

Tag Renaming Attribute	**Triggers on variants that...**
`RenameStrand=`*`tag2`*	Have a strand reversal
`RenameRefalt=`*`tag2`*	Have a REF⇆ALT switch
`RenameTlafer=tag2`	Have both a strand reversal and REF⇆ALT switch
`RenameAlways=tag2`	All variants

If a variant contains a ##INFO or ##FORMAT meta-information line with ID=*tag* and one or more `Rename*` attributes, and the trigger of that tag-renaming attribute occurs, then the tag itself rather than the value (as in *tag*=*value* in an INFO field, or the tag name appearing in the FORMAT field) is changed to *tag2*.

If the RendAlg attribute appears in addition to a Tag Renaming attribute, then both are applied.

If both `RenameStrand` and `RenameRefalt` are specified, then `RenameTlafer` *must* be specified too, and conversely, if `RenameTlafer` is specified, then both `RenameStrand` and `RenameRefalt` *must* be specified too. If `RenameAlways` is specified, other tag renaming attributes *must not* be specified.

For each ##INFO or ##FORMAT line *tag1* with a `Rename*` attribute with a value of *tag2*, a corresponding ##INFO or ##FORMAT line must exist with *tag1* and *tag2* interchanged, which has the same `Rename*`, `Number` and `Type` attributes as the *tag1* line.

Examples:

- Switching FORMAT/ADF ⇆ FORMAT/ADR upon strand reversal:

```
##FORMAT=<ID=ADF,Number=R,Type=Integer,Description="Allelic depths on
the forward strand",RendAlg="R”,RenameStrand=”ADR”>
```

Note: Having the corresponding ADR meta-information line as well is required:

```
##FORMAT=<ID=ADR,Number=R,Type=Integer,Description="Allelic depths on
the reverse strand",RendAlg="R",RenameStrand="ADF">
```

- Dropping an annotation INFO/CLNHGVS in the opposite rendition by renaming it to DROP_CLNHGVS:

```
##INFO=<ID=CLNHGVS,Number=1,Type=String,RenameAlways="DROP_CLNHGVS">
```

- Dropping the annotation INFO/MAX_AF in case of a REF⇄ALT switch:

```
##INFO=<ID=MAX_AF,Number=1,Type=Float,RenameRefalt="DROP_MAX_AF">
```

- Handling annotations that are sensitive to both REF⇄ALT switch and a strand reversal:

```
##FORMAT=<ID=REF_F2R1,Number=1,Type=Integer,RenameRefalt="ALT_F2R1",Ren
ameStrand="REF_F1R2",RenameTlafer="ALT_F1R2">
##FORMAT=<ID=REF_F1R2,Number=1,Type=Integer,RenameRefalt="ALT_F1R2",Ren
ameStrand="REF_F2R1",RenameTlafer="ALT_F2R1">
##FORMAT=<ID=ALT_F2R1,Number=1,Type=Integer,RenameRefalt="REF_F2R1",Ren
ameStrand="ALT_F1R2",RenameTlafer="REF_F1R2">
##FORMAT=<ID=ALT_F1R2,Number=1,Type=Integer,RenameRefalt="REF_F1R2",Ren
ameStrand="ALT_F2R1",RenameTlafer="REF_F2R1">
```

6.5. Sorting

The Primary and Luft renditions are both sorted by their respective coordinates, as required by the VCF specification.

Variants appearing in `##primary_only` and `##lift_only` meta-information lines are not required to be sorted.

7. Rejection and reasons

The *Lifter* or *Renderer* may reject a variant, which in effect declares it to be a single-coordinate variant in the current coordinates.

The *Renderer* may also reject a variant that is already a dual-coordinate variant, turning it into a single-coordinate variant. This may happen, for example, if a new INFO or FORMAT field were added that the *Renderer* cannot cross-render.

When cross-rendering, a *Renderer must* either render the entire variant with all fields cross-rendered as specified, or reject the variant. In other words, if there is a field of a variant which the *Renderer* cannot cross-render for any reason - then the entire variant *must* be rejected, and set the DVCF tag to `Lrej` or `Prej` with the *Reason*. If there are multiple *Reasons* for rejection, the implementation must still list just one *Reason*.

An implementation may use the *Reasons* listed in the table, in which case it *must* use them only when the *Occurrence* in the table occurs. It may also use implementation-specific reasons.

	Reason	**Occurrence**
`Mapping reasons`	`ChromNotInPrimReference`	CHROM does not appear in Primary reference file
	`ChromNotInChainFile`	CHROM has no alignment in chain file
	`NoMappingInChainFile`	POS has no alignment in chain file
`RendAlg reasons`	`INFO/`*`tag`*	INFO/*tag* cannot be cross-rendered
	`FORMAT/`*`tag`*	FORMAT/*tag* cannot be cross-rendered
`Other reasons`	`AddedVariant`	When cross-rendering, the variant had no DVCF tag
	`Rejected`	Other rejection reason

Appendix 2: Compression of cancer VCF files

A special case of VCF files are VCF files used in cancer research, usually containing 2 samples from the same individual - one sample coming for a normal cell and the other from the tumour.

An assessment of Genozip's capability to compress such files was performed on the file HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.vcf obtained from https://www.nygenome.org/bioinformatics/3-cancer-cell-lines-on-2-sequencers/. The result hereinafter shows the Genozip compressed this file by a factor of 13.7, with 52.9% of the information content of the compressed file being the CSQ ("consequences") field. In comparison, bgzip (which implements the gzip algorithm), compresses by a factor 6.9 and bcftools (used to generate a .bcf file) by a factor of 6.1.

```
> ls -lGU HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.*
-rw-rw-r--+ 1 a1786210 1173619 Oct  7 17:09 HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.bcf
-rw-rw-r--+ 1 a1786210 7116918 Oct  7 17:01 HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.vcf
-rw-rw-r--+ 1 a1786210  521328 Oct  6 22:47 HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.vcf.genozip
-rw-rw-r--+ 1 a1786210 1030024 Oct  7 17:01 HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.vcf.gz
```

```
Sections (sorted by % of genozip file):
NAME                   GENOZIP      %        TXT      %   RATIO
INFO/CSQ              267.2 KB  52.9%     2.5 MB  36.4%    9.5X
POS                    69.5 KB  13.8%   239.9 KB   3.5%    3.5X
FORMAT/AD              59.5 KB  11.8%   231.1 KB   3.3%    3.9X
FORMAT/AF              45.1 KB   8.9%   174.9 KB   2.5%    3.9X
REF+ALT                19.5 KB   3.9%   125.5 KB   1.8%    6.5X
TXT_HEADER             18.1 KB   3.6%   175.6 KB   2.5%    9.7X
INFO/CancerGeneCensu    6.3 KB   1.3%   359.4 KB   5.2%   56.8X
INFO/called_by          4.3 KB   0.9%   398.4 KB   5.7%   91.9X
INFO/num_callers        3.2 KB   0.6%    26.0 KB   0.4%    8.0X
ID                      3.0 KB   0.6%    58.6 KB   0.8%   19.5X
INFO                    2.4 KB   0.5%     1.7 MB  24.9%  715.2X
FORMAT/DP               2.3 KB   0.5%   110.8 KB   1.6%   48.1X
INFO/TYPE               1.7 KB   0.3%    77.9 KB   1.1%   46.2X
Other                   1.3 KB   0.3%   155.0 KB   2.2%  117.9X
CHROM                    418 B   0.1%   140.3 KB   2.0%  343.8X
FORMAT                   390 B   0.1%   232.3 KB   3.3%  609.9X
RandomAccessIndex        322 B   0.1%          -   0.0%    0.0X
INFO/supported_by        216 B   0.0%     4.4 KB   0.1%   21.1X
FORMAT/PS                 45 B   0.0%   129.8 KB   1.9%   2953X
QUAL                      42 B   0.0%    51.9 KB   0.7%   1266X
FORMAT/.                  42 B   0.0%      338 B   0.0%    8.0X
INFO/HighConfidence       41 B   0.0%          -   0.0%    0.0X
TOTAL                 505.0 KB 100.0%     6.8 MB 100.0%   13.8X
```

Appendix 3: List of institutions in which Genozip is being used

This list, current as of April 2022, is based on the registration form users are required to complete when they use Genozip for the first time. This is a partial list, including only academic and public institutions. Companies and private hospitals using Genozip were omitted for privacy reasons. The list is kept current on https://genozip.com/institutions.html.

Argentina
Universidad Nacional de Entre Ríos

Australia
University of Adelaide
The University of New South Wales - Sydney
Griffith University
University of Tasmania
Flinders University
Peter MacCallum Cancer Centre
South Australian Health and Medical Research Institute
Torrens University
Victorian Clinical Genetics Services
Telethon Kids Institute
PathWest

Belgium
Royal Museum of Central Africa
Université libre de Bruxelles
Biobix

Brazil
Brazilian Agricultural Research Corporation

Canada
McGill University
Langara College

Chile
Universidad de Magallanes

China
Shanghai Cancer Institute
Institute of Hematology & Blood Diseases Hospital
Oil Crops Research Institute
Yangzhou University
Nanjing University of Posts and Telecommunications
South China Agricultural University
Northwest A&F University
Hunan University
Shanghai Center for Plant Stress Biology
Chongqing Medical University
Nanjing Medical University
Institute of Hematology & Blood Diseases Hospital, Chinese Academy of Med
Shenzhen University
Tsinghua University
South China Normal University
Huazhong Agricultural University
Capital Normal University
Sichuan University
Shanghai Jiao Tong University

Czechia

Czech Technical University in Prague

Anne's University Hospital in Brno

Denmark

University of Copenhagen

Estonia

University of Tartu

France

National Research Institute for Agriculture, Food and Environment (INRAE)

University of Lille

Centre national de la recherche scientifique (CNRS)

Inserm

GenHotel

Assistance Publique - Hopitaux de Paris

Germany

Pediatric Oncology University Hospital Düsseldorf

University Hospital Heidelberg

Max Planck Institute for Chemical Ecology

Max Planck Institute for Plant Breeding Research

Universtatsklinikim Schleswig-Holstein

GEOMAR Helmholtz-Zentrum für Ozeanforschung Kiel

Heinrich Heine University Düsseldorf

Leibniz Institute for the analysis of Biodiversity Change

Greece

Institute of Molecular Biology and Biotechnology-FORTH

Hungary

ELKH Centre for Agricultural Research

India

Regional Centre for Biotechnology

Institute of Life Science

Yenepoya University

Indonesia

Eijkman Institute

Israel

Tel Aviv University

Italy

University of Naples

Japan

Kyoto University

National Cancer Center Research Institute

Shizuoka Cancer Center

Nagoya University

Tokyo University of Agriculture and Technology

University of Tokyo

Nippon Veterinary and Life Science University

Tokyo Medical and Dental University

Kumamoto University

Human Genome Center

Ehime University

National Institute of Henetics

Tokyo Seiei College

Riken

Korea

Yonsei university

Seoul National University

Seoul National University Hospital

Korea Research Institute of Bioscience and Biotechnology

Ulsan National Institute of Science and Technology

Animal and Plant Quarantine Agency

Ewha Womans University

Sungkyunkwan University

Lithuania

Vilnius University

Luxembourg

Luxembourg Centre for Systems Biomedicine

Malta

University of Malta

Mexico

Universidad Autonoma de Sinaloa

Universidad Nacional Autónoma de México

Netherlands

University Medical Center Utrecht

Delft University of Technology

University Goettingen

Norway

University of Oslo

Poland

Silesian University of Technology

University of Warsaw

Russia

Institute of Chemical Biology and Fundamental Medicine

Federal Research Center for Animal Husbandry

Limnological institute

Singapore

National University of Singapore

National Cancer Centre Singapore

South Africa

University of Bayreuth

University of Witwatersrand

Spain

Spanish National Cancer Research Center

Centre for research in agricultural Genomics

Institut Hospital del Mar d'Investigacions Mèdiques

Sweden

Uppsala University

Swedish National Genomics Infrastructure

University of Jyväskylä

Gothenburg University

Thailand

Mahidol University

Siriraj hospital

Taiwan

National Taiwan University

Turkey

Middle East Technical University

Hacettepe University

United Kingdom

University of Edinburgh

Wellcome Sanger Institute

University College London

University of East Anglia

University of Liverpool

United States of America

University of California San Diego

University of Michigan

National Institute of Child Health and Human Development

University of Miami

Duke University

Iowa State University

Beth Israel Deaconess Medical Center

Auburn University

Vanderbilt University

Stanford University

Brown University

University of Wisconsin-Madison

University of Nevada, Las Vegas

Brigham Young University

University of North Texas

University of South Carolina

University of California San Francisco

Columbia University

Montana State University

Emory University

Cornell University

Harvard University

University of California Santa Barbara

Wistar Institute

Scripps Research