Compression of DNA Sequencing Data

Von der Fakultät für Elektrotechnik und Informatik

der Gottfried Wilhelm Leibniz Universität Hannover

zur Erlangung des akademischen Grades

Doktor-Ingenieur

(abgekürzt: Dr.-Ing.)

genehmigte

Dissertation

von

Dipl.-Ing. Jan Voges

geboren am 22. Juni 1988 in Hannover

2022

Referent:	Prof. Dr.-Ing. Jörn Ostermann
Korreferentin:	Prof. Dr. Idoia Ochoa Alvarez
Vorsitzender:	Prof. Dr.-Ing. Bodo Rosenhahn

Tag der Promotion: 13. Mai 2022

Für

Dani
Freddie
Floki

VORWORT

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Informationsverarbeitung der Leibniz Universität Hannover.

Herrn Professor Dr.-Ing. Jörn Ostermann danke ich für die Anregung zum Thema meiner Arbeit und für die Übernahme des Hauptreferats. Ihm gilt mein besonderer Dank für die exzellente Betreuung, hervorragende Arbeitsmöglichkeiten, gehaltvolle Diskussionen, sowie zahlreiche Ideen und Denkanstöße.

Frau Professor Dr. Idoia Ochoa Alvarez danke ich herzlich für die Übernahme des Korreferats.

Herrn Professor Dr.-Ing. Bodo Rosenhahn danke ich für viele ideenreiche Gespräche und Anregungen sowie für die Übernahme des Vorsitzes der Prüfungskommission.

All meinen Kolleginnen und Kollegen, die durch Diskussionen, Impulse und vielerlei praktische Unterstützung zum Gelingen der Arbeit beigetragen haben, danke ich herzlich.

Besonders danken möchte ich an dieser Stelle meiner Frau Daniela für die von ihr aufgebrachte immense Unterstützung. Diese Arbeit widme ich unserem Sohn Freddie.

ABSTRACT

With the release of the latest generations of sequencing machines, the cost of sequencing a whole human genome has dropped to less than US$1,000. The potential applications in several fields lead to the forecast that the amount of DNA sequencing data will soon surpass the volume of other types of data, such as video data. In this dissertation, we present novel data compression technologies with the aim of enhancing storage, transmission, and processing of DNA sequencing data.

The first contribution in this dissertation is a method for the compression of aligned reads, i.e., read-out sequence fragments that have been aligned to a reference sequence. The method improves compression by implicitly assembling local parts of the underlying sequences. Compared to the state of the art, our method achieves the best trade-off between memory usage and compressed size.

Our second contribution is a method for the quantization and compression of quality scores, i.e., values that quantify the error probability of each read-out base. Specifically, we propose two Bayesian models that are used to precisely control the quantization. With our method it is possible to compress the data down to 0.15 bit per quality score. Notably, we can recommend a particular parametrization for one of our models which—by removing noise from the data as a side effect—does not lead to any degradation in the distortion metric. This parametrization achieves an average rate of 0.45 bit per quality score.

The third contribution is the first implementation of an entropy codec compliant to MPEG-G. We show that, compared to the state of the art, our method achieves the best compression ranks on average, and that adding our method to CRAM would be beneficial both in terms of achievable compression and speed.

Finally, we provide an overview of the standardization landscape, and in particular of MPEG-G, in which our contributions have been integrated.

Keywords: compression, DNA sequencing, MPEG-G

KURZFASSUNG

Mit der Einführung der neuesten Generationen von Sequenziermaschinen sind die Kosten für die Sequenzierung eines menschlichen Genoms auf weniger als 1.000 US-Dollar gesunken. Es wird prognostiziert, dass die Menge der Sequenzierungsdaten bald diejenige anderer Datentypen, wie z.B. Videodaten, übersteigen wird. Daher werden in dieser Arbeit neue Datenkompressionsverfahren zur Verbesserung der Speicherung, Übertragung und Verarbeitung von Sequenzierungsdaten vorgestellt.

Der erste Beitrag in dieser Arbeit ist eine Methode zur Komprimierung von alignierten Reads, d.h. ausgelesenen Sequenzfragmenten, die an eine Referenzsequenz angeglichen wurden. Die Methode verbessert die Komprimierung, indem sie die Reads nutzt, um implizit lokale Teile der zugrunde liegenden Sequenzen zu schätzen. Im Vergleich zum Stand der Technik erzielt die Methode das beste Ergebnis in einer gemeinsamen Betrachtung von Speichernutzung und erzielter Komprimierung.

Der zweite Beitrag ist eine Methode zur Quantisierung und Komprimierung von Qualitätswerten, welche die Fehlerwahrscheinlichkeit jeder ausgelesenen Base quantifizieren. Konkret werden zwei Bayes'sche Modelle vorgeschlagen, mit denen die Quantisierung präzise gesteuert werden kann. Mit der vorgeschlagenen Methode können die Daten auf bis zu 0,15 Bit pro Qualitätswert komprimiert werden. Besonders hervorzuheben ist, dass eine bestimmte Parametrisierung für eines der Modelle empfohlen werden kann, die – durch die Entfernung von Rauschen aus den Daten als Nebeneffekt – zu keiner Verschlechterung der Verzerrungsmetrik führt. Mit dieser Parametrisierung wird eine durchschnittliche Rate von 0,45 Bit pro Qualitätswert erreicht.

Der dritte Beitrag ist die erste Implementierung eines MPEG-G-konformen Entropie-Codecs. Es wird gezeigt, dass der vorgeschlagene Codec die durchschnittlich besten Kompressionswerte im Vergleich zum Stand der Technik erzielt und dass die Aufnahme des Codecs in CRAM sowohl hinsichtlich der erreichbaren Kompression als auch der Geschwindigkeit von Vorteil wäre.

Abschließend wird ein Überblick über Standards zur Komprimierung von Sequenzierungsdaten gegeben. Insbesondere wird hier auf MPEG-G eingangen, da alle Beiträge dieser Arbeit in MPEG-G integriert wurden.

Stichworte: Kompression, DNA-Sequenzierung, MPEG-G

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION

Here we provide a concise reference describing the notation throughout this dissertation. A similar notation is used in [GBC16].

NUMBERS AND ARRAYS

a	A scalar
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable
$(a_0\ a_1 \cdots a_{n-1})^\top$	A vector consisting of the n elements a_0 to a_{n-1}

INDEXING

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 0
$A_{i,j}$	Element i, j of matrix $\mathbf{A}$, with indexing starting at 0

SETS

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of integers
$\mathbb{N}$	The set of natural numbers, including 0
$\lvert\mathbb{A}\rvert$	The cardinality of set $\mathbb{A}$
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \ldots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \setminus \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\{x \in \mathbb{R} \mid x > 0\}$	Set-builder notation: the set of all strictly positive real numbers

FUNCTIONS

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f(x)\big\vert_{x=0}$	Value obtained by evaluating the function f at $x = 0$
$f(x; y)$	A function f of x parametrized by y

PROBABILITY AND INFORMATION THEORY

$P(a)$	A probability distribution over a discrete random variable
$p(a)$	A probability distribution over a continuous random variable
$a \sim P$	Random variable a has distribution P
$\mathbb{E}_{a\sim P}[f(a)]$	Expectation of $f(a)$ with respect to $P(a)$
$\mathrm{var}_{a\sim P}(f(a))$	Variance of $f(a)$ under $P(a)$
$\mathrm{cov}_{a\sim P}(f(a), g(a))$	Covariance of $f(a)$ and $g(a)$ under $P(a)$

ACRONYMS

AG advisory group. 114
AliCo alignment compressor. 7, 8, 124
ANS asymmetric numeral systems. 10, 37, 38, 92, 99, 100, 134
API application programming interface. 114, 115, 118, 126
ASCII American Standard Code for Information Interchange. 5, 20–23, 66, 90
AVC Advanced Video Coding. 42, 125

BGZF blocked gzip file format. 25, 57, 92
bp base pair. 1, 6, 17–20

CABAC context-adaptive binary arithmetic coding. vii, xiii, 10, 37, 42, 92–95, 98, 104, 106, 125, 133, 134
CALQ Coverage-adaptive Lossy Quality Score Compressor. ix, 7–10, 37, 61–64, 77, 82, 84, 86–89, 127, 132–134
CEPH Centre d'étude du polymorphisme humain. 100

DNA deoxyribonucleic acid. vii, ix, 1–11, 13–20, 26, 42, 43, 46, 47, 49, 91–93, 99, 102, 104, 105, 131, 133, 134

EG Exponential Golomb. xiv, xv, 97, 98
EGA European Genome-Phenome Archive. 118, 126

GA4GH Global Alliance for Genomics and Health. 10, 114, 115
GABAC Genomics-oriented CABAC. viii, ix, 7, 8, 10, 11, 37, 91–95, 97–100, 102–107, 110, 111, 125, 127, 133, 134
GATK Genome Analysis Toolkit. 64, 75, 79, 80, 101, 135
Genie Genomic Information Codec. 7, 8, 114, 127, 128, 133
GRC Genome Reference Consortium. xiii
GRCh37 GRC human build 37. 46, 101

HBB hemoglobin subunit beta (also known as beta globin). vii, 21, 22
HEVC High Efficiency Video Coding. 42, 125
HF hard filtering. viii, 135, 138, 141, 144
HGP Human Genome Project. 1, 3
HTS high-throughput sequencing. 1–5, 16, 18, 113

IEC International Electrotechnical Commission. vii–ix, 7, 10, 11, 45, 52–54, 60, 61, 77–80, 88, 91–94, 97, 99, 104, 107, 114–121, 123, 125–129, 131–133
IEEE Institute of Electrical and Electronics Engineers. 42
ISO International Organization for Standardization. vii–ix, 7, 10, 11, 45, 52–54, 60, 61, 77–80, 88, 91–94, 97, 99, 104, 107, 114–121, 123, 125–129, 131–133
IUPAC International Union of Pure and Applied Chemistry. 5, 21, 22

JTC joint technical committee. 114

LUT lookup table. 44, 79, 93–97, 106
LZMA Lempel-Ziv-Markov chain algorithm. 100

MIT master index table. 117, 119
MPEG Moving Picture Experts Group. 114, 116
mRNA messenger RNA. 15
MSE mean squared error. 82–86

NB national body. 93, 104
NCBI National Center for Biotechnology Information. xiv, 21
NGS next-generation sequencing. 1, 3, 18
NIST National Institute of Standards and Technology. 81

OICR Ontario Institute for Cancer Research. 115
ONT Oxford Nanopore Technologies. 17, 19, 20

PCC Pearson correlation coefficient. 31
PCR polymerase chain reaction. 1
PDF probability density function. 28, 29
PMF probability mass function. 28

QS quality score. viii, 83–87, 89, 90, 136–144
QScomp Quality Score Compressor. 7, 8

RAM random-access memory. 59
RNA ribonucleic acid. xiv, 14–16, 55
RSS resident set size. ix, 59, 60

SC subcommittee. 114, 116
SEG signed EG. 97
SMRT single-molecule real-time. 16, 17, 19, 55, 56, 58
SNP single-nucleotide polymorphism. 19, 62
SRA NCBI Sequence Read Archive. 23, 118, 126

STEG signed TEG. 97

TC technical committee. 114, 116
TCGA The Cancer Genome Atlas. 1
TEG truncated EG. xv, 97, 98
TSC TNT Sequence Compressor. ix, 7, 8, 10, 37, 45, 46, 48–55, 57–60, 127, 131–134
TU truncated unary. 97, 98

URL uniform resource locator. 134

VQSLOD variant quality score log-odds. 80
VQSR variant quality score recalibration. viii, 79, 80, 135–137, 139, 140, 142, 143

WG working group. 114
WGS whole genome sequencing. ix, 6, 20, 55, 56, 90, 100, 108, 120, 123
WHO World Health Organization. 115

ZMW zero-mode waveguide. 19

1

INTRODUCTION

In the year 1977, the first full genome was sequenced: it was the genome of bacteriophage ΦX174, consisting of a mere 5,386 base pairs (bp) [San+77]. Since then, the techniques used for DNA sequencing, i.e., the process of determining the sequence of nucleotides in DNA, have made great progress.

DNA is a molecule that carries the genetic information responsible for growth, development, functioning, and reproduction of all organisms and many viruses on earth. In 1953, James Watson and Francis Crick identified the double helix molecular structure of DNA [WC53], depicted schematically in Figure 1.1. Aided by technologies such as PCR, researchers were able to complete the sequencing of more and longer genomes: the genome of bacteriophage λ (48,502 bp) was published in 1982 [San+82], and the genome of *Escherichia coli* (approximately 4.6 million bp) was published in 1997 [Bla+97]. These efforts culminated in the completion of the HGP [Int01]. An initial rough draft of the human genome (approximately 3 billion bp) was published in 2000, and the project was declared complete in 2003.

In the mid to late 1990s, several new methods for DNA sequencing were developed. These were called high-throughput sequencing (HTS) or next-generation sequencing (NGS) methods to distinguish them from the earlier methods. By the year 2000, the first commercial DNA sequencing machines that implemented these new methods were available. In the 21st century, many projects—such as the 1000 Genomes Project [The10], with the aim to create a detailed catalog of human genetic variation, and TCGA[1], targeting genetic mutations responsible for cancer—have been driving the development of, and benefited from, HTS technologies.

The HGP took almost 13 years to complete and it cost approximately US$3 billion; that is roughly US$1 per bp. However, with the introduction

1 https://www.cancer.gov/tcga

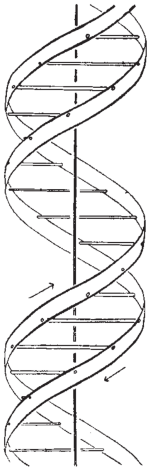


Figure 1.1: The double helix molecular structure of DNA. Reprinted by permission from Springer Nature [WC53].

of the latest generation of sequencing equipment, the cost of sequencing a whole human genome has been reduced to less than US$1,000[2].

The potential applications in several fields—such as precision medicine and oncology—lead to the forecast that the amount of DNA sequencing data will soon surpass the volume of video data uploaded to YouTube, or tweets posted on Twitter [Ste+15]. By that point the costs associated with storing, transmitting, and processing the large volumes of DNA sequencing data will largely exceed the sequencing costs.

The main objective of this dissertation is the development of compression technology that facilitates scalable DNA sequencing data storage, transmission, and access. Consequently, the developed technologies can help create an ecosystem of applications, as well as eventually democratize and fully exploit its yet-to-be-discovered potential.

1.1 MOTIVATION

Nowadays, HTS technologies are used to generate the majority of DNA sequencing data. HTS technologies achieve a high throughput by massively parallel sequencing of DNA fragments, i.e., they do not sequence

2 https://www.genome.gov/sequencingcostsdata

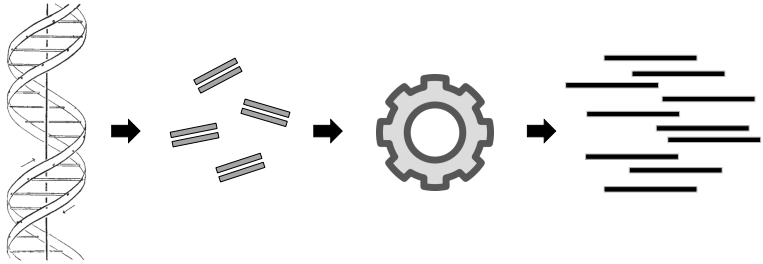


Figure 1.2: DNA sequencing workflow. The input DNA is preprocessed ("library preparation") before the actual sequencing. The sequencing yields a set of reads. Each read corresponds to a single DNA fragment.

entire DNA molecules[3]. A set of preprocessing steps is necessary to achieve a high throughput. These preprocessing steps include the fragmentation of DNA molecules into a platform-specific size range and the ligation of specialized adapters to fragment ends. Also, to achieve the mentioned parallelism, the DNA fragments are duplicated multiple times. These preprocessing steps are summarized under the term "library preparation". The actual sequencing, i.e., the reading of DNA fragments, is then performed by a sequencing machine implementing a specific HTS technology. The output of the sequencing process is a set of sequence "read-outs", or "reads", in short. For each DNA fragment in the library, an estimate for each base ("base call") is produced by the sequencing machine. The reads can be thought of as strings randomly sampled from the DNA molecule(s) that are being sequenced. Typically, each base call is accompanied by a quality score which indicates the confidence in the base call. Figure 1.2 schematically illustrates the DNA sequencing workflow.

During the last two decades DNA sequencing costs decreased rapidly and significantly. Figure 1.3 shows the decrease in DNA sequencing costs per human genome. The sequencing of the human genome in the course of the HGP (1990–2003) cost approximately US$3 billion [Int01]. However, around 2001, the cost was estimated to already have decreased to around US$100 million. Until 2007 this cost dropped by roughly one order of magnitude to approximately US$10 million. Although by the year 2000, the first commercial DNA sequencing machines implementing NGS methods were available, they only saw widespread market adoption around 2007. Consequently, in the following decade the cost

3 So-called "third-generation" sequencing technologies (e.g., nanopore sequencing) are able to sequence entire DNA molecules. However, the data generated by third-generation technologies accounts only for a minor share of the entire DNA sequencing data being generated.

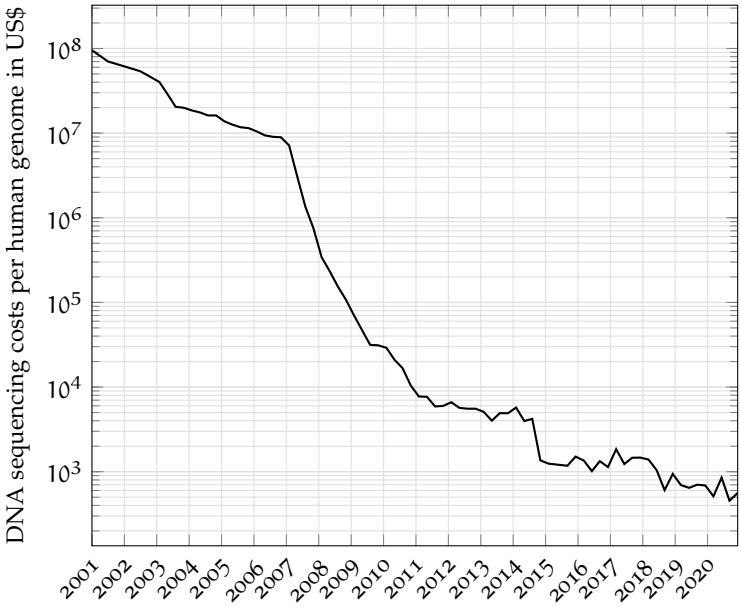


Figure 1.3: DNA sequencing costs per human genome. Data from: `https://www.genome.gov/sequencingcostsdata`.

dropped even more drastically. Finally, with the introduction of the latest generation of sequencing equipment, the cost of sequencing a whole human genome has been reduced to less than US$1,000. The decrease in cost is even outpacing Moore's law. Originally, Moore's law is based on the finding that the number of transistors in a dense integrated circuit doubles approximately every two years. Applied to the subject matter, Moore's law would predict that the cost of the sequencing of a human genome would be halved every two years. While this roughly holds true for the period from 2001 to 2007, the advent of HTS technologies led to a tremendous drop in the cost of DNA sequencing in the period from 2007 to 2009. During these two years the cost of DNA sequencing per human genome dropped by a factor of approximately 40, far outpacing the cost that would have been predicted by Moore's law. The reduced cost facilitates larger research studies, precision medicine, and the commercialization of DNA sequencing. However, this and other trends in HTS data generation indicate that storage, transmission, and bandwidth costs will soon surpass the cost of sequencing and become the main bottleneck

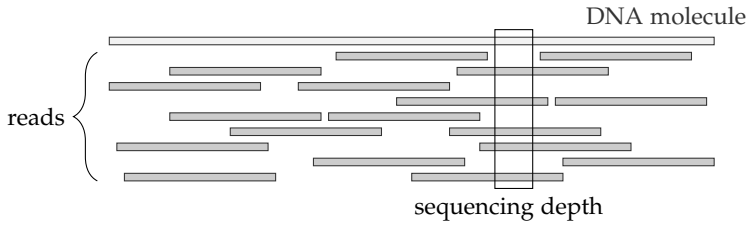


Figure 1.4: Reads, sequencing depth, and coverage. The top of the figure schematically shows a DNA molecule which is subject to sequencing. Below, the set of reads which is generated by the sequencing process is depicted. Here we assume that the reads were produced with a sequencing technology that yields reads with a constant length. A location with a sequencing depth of 5 is highlighted. The coverage is the average sequencing depth over all locations. In this example the coverage is 4.5×.

in omics[4] as well as in the application of HTS data to precision medicine. One approach to mitigate the burden of HTS data on storage, bandwidth, and transmission is the use of high-performance compression techniques specifically designed for HTS data.

Viewed from another angle, the drop in cost of DNA sequencing also facilitates more sophisticated research studies by performing "deeper" sequencing experiments. The sequencing depth indicates how many reads are estimated to overlap a specific location on the DNA molecule that was sequenced. The coverage is the average sequencing depth over all locations. It is usually expressed as for example "30×". Achieving more coverage is costly, since more consumables (for library preparation and operating sequencing machines) must be accounted for. However, in most research settings higher coverage is desirable as it facilitates higher certainties about the underlying DNA molecules. Figure 1.4 illustrates the relations between reads and sequencing depth as well as coverage.

Reads as output by a sequencing machine are typically stored in the FASTQ format [Coc+10]. The FASTQ format is a textual file format where each base is stored as an 8-bit ASCII character. Usually, the characters A, C, G, and T are used for the bases adenine, cytosine, guanine, and thymine, respectively. Also, in the case that a particular base could not be identified ("called"), the character N is typically used. In general, any character from the IUPAC nucleotide codes could be used. Also, each base is accompanied by a quality score, a value indicating the confidence in the base call. Each quality score is also stored as 8-bit ASCII character.

4 The term "omics" encompasses various biology disciplines whose names end in the suffix "-omics", such as genomics and proteomics.

Table 1.1: Transmission times of human WGS data. The sizes are approximate and correspond to the sequencing of a whole human genome at a coverage of 200×.

Contents	Format	Size	Bandwidth	
			25 Mbit/s	1 Gbit/s
Reads	FASTQ	≈ 1,285 GiB	5.1 days	3.1 hours
Alignments	SAM	≈ 2,570 GiB	10.2 days	6.2 hours

The FASTQ format also includes a string identifying each read. We refer the reader to Section 2.3.2 for a more detailed description of the FASTQ format.

Thus, storing the human genome—which consists of roughly 3 billion bp—in the FASTQ format would yield a file with a size[5] of approximately 6.4 GiB. Sequencing the human genome at a coverage of 200× would hence yield a FASTQ file with a size of about 1,285 GiB. Typically, reads are subsequently aligned (e.g., with respect to an externally provided reference genome). This process yields so-called alignments, which are typically stored in the SAM format [Li+09]. Similar to the FASTQ format, the SAM format is a textual file format. With regard to the data in a FASTQ file, the alignments contain additional alignment information as well as further metadata. We refer the reader to Section 2.3.3 for a more detailed description of the SAM format. The additional alignment information and metadata renders SAM files roughly twice as large as their corresponding FASTQ counterparts. To illustrate the impact of these file sizes, Table 1.1 shows example transmission times of human WGS data stored in the FASTQ and SAM formats. As shown in Table 1.1, transferring the SAM file over a network with a bandwidth of 25 Mbit/s takes 10.2 days. However, research studies nowadays are rarely limited to one individual. Typically, it is necessary to exchange DNA sequencing data equivalent to hundreds or thousands of human genomes. Therefore, nowadays, transmission by courier is still a very popular option. It is evident that either significantly higher network bandwidths or more efficient compression technologies are required to satisfy requirements on storage, bandwidth, and transmission.

5 We assume that the strings that identify the reads induce an overhead of 15% with respect to the size that is occupied by bases and quality scores. Hence, here, the approximate FASTQ file size is calculated as $3 \cdot 10^9 \cdot 2 \cdot 1.15\ \text{B} \approx 6.4\ \text{GiB}$.

1.2 STATE OF THE ART AND CONTRIBUTIONS

We present the state of the art in DNA sequencing data compression by dividing it into four different categories: i) the compression of aligned reads, where several specialized compression methods have been proposed in the literature that leverage alignment information to exploit the redundancy introduced by the coverage; ii) the compression of quality scores, where quantization is the key to achieve compression while at the same time preserving downstream analysis performance; iii) the entropy coding of DNA sequencing data, where the goal is to find models that best describe the data; and iv) standards and implementations, where the aim is to provide perennial specifications to enable the widespread adoption of compression technologies to democratize the use of DNA sequencing data. For each category, we point out where our contributions provide improvements over the state of the art.

Figure 1.5 shows the timeline of the state of the art and contributions. Key contributions are shown in bold; supplementary contributions are shown in italics. Several specialized compression methods for the compression of aligned reads (BAM [Li+09], CRAM [Fri+11], Quip [Jon+12], DeeZ [HNS14]) predate our contribution, TSC [VMO16]. Our contributions QScomp [Vog+18b; Vog+18a] and CALQ [VOH17; VOH18] for the compression of quality scores are the most recent methods in a line of specialized quality score compression methods (P-/R-Block [CMT14], QVZ [Mal+15], Quartz [Yu+15], QVZ2 [HOW16]). We also integrated CALQ in AliCo [Och+19], a compressor for aligned reads which is focused on streaming of compressed data. Also, Figure 1.5 shows our contribution for the entropy coding of DNA sequencing data, GABAC [Par+19; Vog+20]. Finally, in Figure 1.5, we show our software Genie, the first open-source implementation of the file format and compression technology specified in the ISO/IEC 23092 series.

1.2.1 *Compression of Aligned Reads*

The first DNA-related compression solutions aimed at the compression of entire genomes. An example is DNACompress [Che+02]. In the years up to 2007 this area was limited to the compression of genomes by means of dictionary-based compression methods. These methods were specifically tailored to the properties (such as approximate repeats and palindromes) of genomes. Starting in 2009, tools were developed that encode genomes with the help of a reference genome. Examples include DNAzip [Chr+09] and iDoComp [OHW15]. With these reference-based

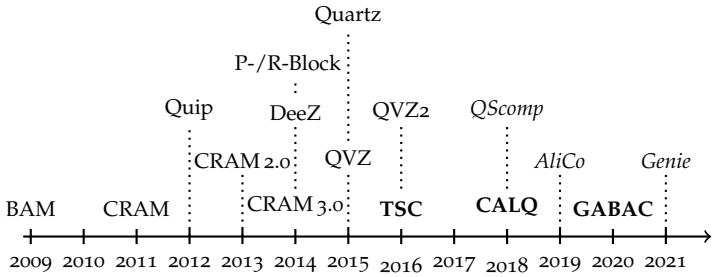


Figure 1.5: Timeline of the state of the art and contributions. Key contributions are shown in bold; supplementary contributions are shown in italics.

methods it is possible to compress a single human genome down to a few megabytes.

The compression of reads differs significantly from the compression of entire genomes, since a set of reads exhibits intrinsic redundancy, due to the coverage. Tools for the compression of reads are for example SCALCE [Hac+12] and ORCOM [GDR15]. ORCOM frequently yields the best compression, since it changes the order of reads in such a way that compression efficiency is maximized. Changing the order of reads is a legitimate approach because the reads constitute a set; in principle they are randomly sampled fragments from the underlying DNA molecules.

Aligned reads are commonly stored in the SAM format [Li+09]. Due to the alignment information (and the sorting of alignments by their loci), the redundancy introduced by the coverage is easily accessible for compression algorithms. To address the compression of aligned reads, several specialized compression methods have been proposed in the literature. Examples are BAM [Li+09], CRAM [Fri+11] (in its different versions), Quip [Jon+12], and DeeZ [HNS14]. These methods use reference sequences for the compression of aligned reads, in combination with dictionary-based coding or arithmetic coding. Currently, CRAM 3.0 is the method seeing the broadest acceptance. Compared to its predecessor CRAM 2.0, CRAM 3.0 generally provides better compression by including more sophisticated entropy codecs. However, in one of our previous works we have shown that newer methods achieve better compression than both CRAM 2.0 and CRAM 3.0 [Num+16].

Our contribution, TSC [VMO16], was designed with the goals of achieving a low memory footprint and operating without external reference sequences. It combines alignment information to implicitly assemble local parts of the underlying genome to compress alignments. In contrast to the state of the art, the proposed method does not require reference

sequences to encode alignments. Compression is performed on-the-fly using only a sliding window (i.e., a continuously updated short-time memory) as the context for predicting reads. This eliminates the need to hold reference sequence(s) readily available in memory.

1.2.2 *Compression of Quality Scores*

None of the above solutions for genome, read, or alignment compression focuses primarily on the compression of quality scores. However, it has been shown that quality scores can take up to 80% of the lossless compressed size [Och+17]. To further reduce file sizes, Illumina proposed a binning method to reduce the number of different quality scores from 42 to 8. With this proposal, Illumina opened the door for lossy compression of quality scores.

The disadvantage of lossy compression of quality scores is that downstream analyses could be affected by the loss incurred with this type of compression. However, Yu et al. [Yu+15], Ochoa et al. [Och+17], as well as our own works [Alb+16; Her+17; Her+18] showed that quality scores compressed with more advanced methods can not only perform better in downstream analyses than Illumina-binned quality scores, but in some cases even better than the original quality scores, because these methods remove noise from the data. These conclusions were made by conducting rate-distortion analyses, where novel distortion metrics are applied that are specifically tailored to different DNA sequencing data workflows such as variant calling.

Our contribution to the field of compression of quality scores is centered around minimizing the impact of lossy quality score compression on downstream analyses. All the previously proposed lossy compressors for quality scores are primarily focused on (unaligned) reads; and even if those compressors could be easily applied to alignments, they do not exploit the extra alignment information. Therefore, it is of primary importance to propose a specialized method for aligned data. Our contribution, CALQ [VOH17; VOH18], exploits the alignment information to build sophisticated models to measure the "activity" at each locus of the underlying genome. CALQ further determines the acceptable level of distortion for the quality scores such that subsequent downstream analyses are presumably not affected. Finally, the quality scores are quantized and compressed accordingly. In this way, a high compression is achieved with a negligible impact on downstream analyses.

1.2.3 *Entropy Coding of DNA Sequencing Data*

The first step of typical state-of-the-art DNA sequencing data compressors such as CRAM [Fri+11] or DeeZ [HNS14] is to represent the information by splitting the data into a set of homogeneous descriptor streams, where each contains a specific type of data (e.g., mapping positions, quality scores, or mismatch information). Each descriptor stream is more likely to contain stationary data, enabling a more efficient compression. However, the state-of-the-art methods used to compress such streams are general-purpose compression methods. CRAM uses a mixture of experts comprising the general-purpose compressors gzip, bzip2, xz, and a range variant of the family of ANS methods [Dud14]. DeeZ uses a combination of dictionary-based and arithmetic coding-based compression methods.

Our contribution, GABAC [Par+19; Vog+20], combines proven coding technologies, such as CABAC [MSW03], binarization schemes, and transformations, into a straightforward solution for the compression of DNA sequencing data.

1.2.4 *Standards and Implementations*

The growth of the DNA sequencing market in recent years prompted the development of standardized data formats for DNA sequencing data. Over the last years, besides growing activities in the vibrant bioinformatics research community, there has been a proliferation of commercial initiatives, such as 23andMe[6] and MyHeritage[7], as well as government-led initiatives targeting the application of DNA sequencing on population scale. On the forefront of standardized data formats is the ISO/IEC 23092 series (branded as MPEG-G), a standard series developed by ISO and IEC [VO17; Alb+18]. Most technologies presented in this dissertation were submitted to the standardization process of the ISO/IEC 23092 series. However, ISO and IEC are not the only organizations standardizing DNA sequencing data compression technologies. Most notable, founded in 2013, GA4GH is an international coalition focused on implementing effective genomic and clinical data sharing. Both standardization efforts have been started following industry and market needs.

Most of our contributions in the above-mentioned fields are adopted in the ISO/IEC 23092 series. Most prominently, the technology behind our contribution TSC [VMO16; MVO] as well as the technology behind our contribution CALQ [VOH17; VOH18; VHO; VOb; VOa] were adopted.

6 https://www.23andme.com

7 https://www.myheritage.com

1.3 OUTLINE

In Chapter 2, we introduce the fundamentals that are necessary for the understanding of this dissertation. These fundamentals range from molecular biology over a detailed essay about DNA sequencing technologies to data compression. We also provide a brief introduction to the field of bioinformatics, to bridge the gap between the classical domains covered by this dissertation: biology on the one hand, and computer science as well as electrical engineering on the other hand.

We present our methods for the compression of aligned reads and quality scores in Chapters 3 and 4. Both chapters include a detailed review of the corresponding state of the art.

In Chapter 5, we investigate entropy coding of DNA sequencing data. Here we discuss the premier implementation of an entropy codec compliant to ISO/IEC 23092-2: GABAC.

Chapter 6 gives an overview of de-facto, industry and international standards for DNA sequencing data representation before focusing on the ISO/IEC 23092 series. We show how our contributions in the abovementioned fields are adopted in the ISO/IEC 23092 series and in which regards they constitute key components. Finally, we present an implementation of the file format and compression technology specified in the ISO/IEC 23092 series.

2

PRELIMINARIES

In this chapter we lay out the foundations necessary for the understanding of the methods presented in Chapters 3, 4, and 5 in this dissertation. These foundations span multiple disciplines. They range from molecular biology in Section 2.1 and DNA sequencing in Section 2.2 over bioinformatics in Section 2.3 to data compression in Section 2.4.

2.1 THE CENTRAL DOGMA OF MOLECULAR BIOLOGY

DNA is the molecule that is the foundation of all organisms and many viruses[1] on earth. It contains the genetic information that is responsible for the functioning of these entities. In 1953, James Watson and Francis Crick identified the double helix molecular structure of DNA [WC53]. This structure is schematically depicted in Figure 1.1 (page 2). The double helix is formed by two strands that coil around each other. Each strand consists of atomic units called nucleotides[2]. Each nucleotide is composed of the monosaccharide[3] deoxyribose, a phosphate group, and a nucleobase (or base, in short). The base can either be adenine (A), cytosine (C), guanine (G), or thymine (T). The deoxyribose of one nucleotide and the phosphate group of the next nucleotide are joined to one another by covalent bonds, forming a so-called sugar-phosphate backbone. The bases of the two strands are connected to one another by hydrogen bonds, according to the base pairing rules (A with T and C with G). Two such complementary bases form a base pair. Hence, the two strands carry the same genetic information, which is encoded in the particular sequence of nucleotides.

1 Organisms are individual entities that embody the properties of life, i.e., metabolism, growth, reproduction, and others. Viruses are commonly not considered as organisms, since they do not have their own metabolism.

2 Technically, a DNA strand is a polymer. A polymer is a large molecule composed of repeated atomic units called monomers. In the case of DNA the nucleotides take the role of monomers.

3 Monosaccharides are also called simple sugars. Beside deoxyribose, examples of monosaccharides include glucose and fructose.

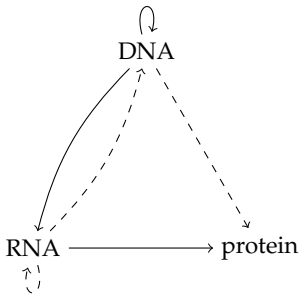


Figure 2.1: The central dogma of molecular biology [Cri58; Cri70]. The three general genetic information transfers are depicted with solid arrows. The three special transfers are shown with dashed arrows.

DNA is not the only molecule capable of carrying genetic information. In general, genetic information can be carried by nucleic acids (i.e., DNA and RNA) or proteins. The genetic information is encoded in the particular sequence of nucleotides, in the case of nucleic acids, or amino acids, in the case of proteins. The flow of genetic information in biological systems between these types of molecules is explained by the central dogma[4] of molecular biology, developed by Francis Crick. He published it in 1958 [Cri58] and refined it in 1970 [Cri70]. The central dogma states that the transfer of genetic information from nucleic acid to nucleic acid, or from nucleic acid to protein may be possible, but that the transfer of genetic information from protein to protein, or from protein to nucleic acid is impossible. More specifically, there are $3^2 = 9$ conceivable types of genetic information transfer between these three molecule types. Of these, three "general" transfers occur normally in most cells: DNA replication (DNA→DNA), transcription (DNA→RNA), and translation (RNA→protein). Three "special" transfers occur only under specific conditions: RNA replication (RNA→RNA), reverse transcription (RNA→DNA), and direct translation from DNA to protein (DNA→protein). The remaining three transfers (protein→protein, protein→DNA, protein→RNA) are believed never to occur. The central dogma of molecular biology is visualized in Figure 2.1.

In what follows we focus on two general genetic information transfers: transcription (DNA→RNA) and translation (RNA→protein), because these two transfers play the major role in gene expression, the process by which genetic information contained in a gene, i.e., a sequence of DNA

4 The use of the word "dogma" in this context is controversial, as also acknowledged by Crick [Cri88]. He suggests that "hypothesis" could be used as well.

nucleotides, is used to synthesize a gene product. Gene products are typically proteins but can also be various types of so-called functional RNA in the case of non-protein-coding genes. In the following description of transcription and translation we focus on the synthesis of proteins.

During transcription, a DNA strand is used as template for the production of a complementary RNA strand[5]. This process is performed by the enzyme RNA polymerase. The RNA strand synthesized by RNA polymerase is called primary transcript. In the case of prokaryotic genes the primary transcript is mRNA, and it is in unaltered form ready for translation into protein. In the case of eukaryotic genes the primary transcript must undergo further processing (yielding "mature" RNA) before it is ready for translation into protein.

During translation an mRNA strand is used as template for the production of one or more proteins. Here, sequences of nucleotide triplets (called codons) are translated into sequences of amino acids, according to the genetic code[6]. These sequences of amino acids further fold, yielding biologically functional proteins with a three-dimensional structure.

Considering the essential role that nucleic acids play in molecular biology, it becomes clear that gaining knowledge about their exact nucleotide sequences is of utmost importance. Note that, as RNA is less stable than DNA, it is technically more feasible to determine DNA nucleotide sequences. The procedure for determining the exact sequence of nucleotides in DNA is known as DNA sequencing. By sequencing DNA it is possible to determine the nucleotide sequences of specific genes, larger stretches of DNA, chromosomes, and thus even entire genomes. DNA sequencing also facilitates the indirect determination of the sequences of the other molecules that are capable of carrying genetic information: RNA (for this purpose RNA is reverse transcribed to complementary DNA) and proteins (accomplished by lookup in databases generated through various DNA sequencing projects, such as UniProt[7]). DNA sequencing has thus emerged as a pivotal technology in many areas of biology and other sciences such as medicine; it has even prompted the advent of new disciplines, of which the most prominent ones are bioinformatics and computational biology[8].

5 Note that in RNA, with respect to DNA, thymine (T) is replaced by uracil (U).

6 The genetic code is a set of rules specifying how to translate genetic information in the form of codons from DNA or mRNA into proteins. For more detailed information we refer the reader to [Lod+07].

7 `https://www.uniprot.org`

8 Bioinformatics and computational biology are related but distinct disciplines. Where bioinformatics refers to the computational study of large sets of biological data, the focus of computational biology is on finding computational solutions to particular biological problems.

2.2 DNA SEQUENCING

DNA sequencing is the process of determining the exact order of nucleotides in DNA. DNA sequencing also facilitates the indirect determination of the sequences of RNA and proteins. In general, DNA sequencing is based on the determination, i.e., reading, of the nucleotide sequences of DNA fragments. Hence, the output of sequencing is a set of sequence "read-outs", or "reads".

Since the 1970s, an abundance of sequencing methods has been developed. They can be classified into "basic" methods (Maxam-Gilbert sequencing, chain-termination sequencing) and HTS methods, which again can be subdivided into short-read methods (massively parallel signature sequencing, polony sequencing, pyrosequencing, combinatorial probe anchor synthesis, sequencing by oligonucleotide ligation and detection, Ion Torrent semiconductor sequencing, DNA nanoball sequencing, Illumina sequencing) and long-read methods (SMRT sequencing, nanopore sequencing) [Qua+12; Liu+12; Chi+13].

Table 2.1 gives an overview about the main characteristics of chain termination sequencing, representing the basic methods, sequencing by synthesis (i.e., Illumina sequencing), representing short-read methods, and SMRT sequencing as well as nanopore sequencing, both representing long-read methods.

2.2.1 *Basic Methods*

In 1977, Frederick Sanger, who also led the effort of sequencing the first full genome—the one of bacteriophage ΦX174 [San+77]—developed, together with colleagues, the first rapid sequencing method: the "chain termination" method for sequencing DNA molecules. He was drawing on results from Padmanabhan, Wu, and colleagues [PW72; RTP73; Jay+74], and published the method in 1977 [SNC77]. This was a major breakthrough and it earned him his second[9] Nobel Prize in Chemistry in 1980 (he and Walter Gilbert shared half of the prize "for their contributions concerning the determination of base sequences in nucleic acids" [Nob21b] with Paul Berg "for his fundamental studies of the biochemistry of nucleic acids, with particular regard to recombinant-DNA" [Nob21b]). During that time, Allan Maxam and Walter Gilbert also developed sequencing methods, including one for "DNA sequencing by chemical degradation" [MG77]. Chain-termination and Maxam-Gilbert sequencing are referred to as "basic" sequencing methods.

9 In 1958, he was awarded a Nobel Prize in Chemistry "for his work on the structure of proteins, especially that of insulin" [Nob21a].

Table 2.1: Comparison of DNA sequencing technologies. Short-read methods (represented by "sequencing by synthesis") provide major improvements with respect to chain termination sequencing. Third-generation methods (SMRT sequencing, nanopore sequencing) provide particularly long reads. Data from [Qua+12], [Liu+12], and [Chi+13]. Regarding the achievable read length with nanopore sequencing: the reading of up to 2,272,580 bp has been reported [Pay+19].

Method	Read Length	Single-Read Accuracy	Cost (per 1 billion bp)
Chain termination sequencing (Sanger)	400–900 bp	99.9%	> US$2 million
Sequencing by synthesis (Illumina)	50–600 bp	99.9%	US$5–150
SMRT sequencing (Pacific Biosciences)	30,000+ bp	~87%	US$5–50
Nanopore sequencing (ONT)	Infinite	92–97%	US$7–100

2.2.2 *High-Throughput Methods*

The soaring demand for low-cost sequencing has prompted the development of HTS methods that parallelize the sequencing process. HTS methods can be subdivided into short-read and long-read methods.

2.2.2.1 *Short-Read Methods*

Process improvements, in particular massive parallelization, led to the second, or next, generation of sequencing methods, called as such to distinguish them from the earlier basic methods. The majority of NGS methods was developed in the mid to late 1990s, and implemented in commercial sequencing machines by the year 2000. Among them, the method used by Illumina, sequencing by synthesis, has been the one seeing the widest use. Sequencing by synthesis is a method that produces rather short reads with lengths between 50 and 600 bp. These short reads typically exhibit a single-read accuracy of more than 99% [GMM16]. Owing to their high throughput, the majority of sequencing data generated today is produced using short-read methods.

In what follows, we briefly revisit the sequencing by synthesis method. This method is composed of three basic steps: sample preparation, cluster generation, and sequencing.

In the sample preparation step, single-stranded DNA is fragmented. Subsequently, adapters are added to the ends of the fragments. The prepared fragments are then loaded onto a flow cell which contains oligonucleotides that provide anchoring points for the adapters.

When the DNA fragments are attached to the oligonucleotides on the flow cell, the cluster generation step takes place. This mainly involves the amplification of DNA fragments, where all copies of a fragment accumulate in the form of a cluster[10]. This guarantees that in the sequencing step, when for a given cluster all identical fragments are read simultaneously, enough light is emitted (from fluorescently tagged nucleotides), such that detection of newly added nucleotides is facilitated.

Next, the sequencing step takes place. Here, complementary strands are synthesized, via DNA polymerase, alongside each (copy of a) DNA fragment. The synthesis is performed in cycles. A cycle begins with modified nucleotides being washed onto the flow cell. These nucleotides are modified in such a way that DNA polymerase can only add one nucleotide at a time to the strands under synthesis. Moreover, these

10 Illumina, for instance, uses so-called "patterned" flow cells that contain nanoscale wells (nanowells) at fixed locations across the flow cell. Clusters can only form in these nanowells. This structured organization of clusters provides several advantages. For instance, it is not necessary to map cluster sites (as the nanowell positions are known).

nucleotides are fluorescently tagged. After the addition of nucleotides, the clusters are excited by a light source and a characteristic fluorescent signal is emitted and captured by a camera. Then, the next cycle may begin. A computer software ("base caller") analyzes the captured images to identify (or "call") the nucleotides. At the end of each cycle, all newly synthesized strands have the same length. Hence, the read length is determined by the number of cycles. As an example, on the Illumina MiSeq sequencer one full cycle takes approximately 6 minutes, and reagent kits for read lengths of 25–300 bp are available. Sequencing at a read length of 150 bp with the "MiSeq Reagent Kit v2 Micro" kit[11] takes approximately 15 hours, plus an overhead of around 4 hours for cluster generation and base calling. In this setting a yield of 8 million reads (i.e., 4 million pairs) would be achieved.

2.2.2.2 *Long-Read Methods*

Sequencing methods producing long reads are often referred to as third-generation methods. They aim at reading the sequences of entire DNA molecules, which can facilitate, among others, a better understanding of large structural variations. While short reads are well suited to discover SNPs, indels, and small structural variations, large structural variations such as copy number variations are difficult—and sometimes impossible—to capture with short-read technology. The most prominent approaches are the SMRT sequencing technology from Pacific Biosciences[12] and nanopore sequencing from ONT[13].

SMRT sequencing technology is based on sequencing by synthesis. However, the reading process is altered such that the reading of much longer DNA molecules becomes practical. Specifically, the reading occurs in tiny ZMWs. A ZMWs can be viewed as a very small light detection volume. Attenuated light illuminates a ZMW from below, penetrating the lower 20–30 nm of each ZMW. A DNA-polymerase complex is immobilized at the bottom of the ZMW and color-labeled nucleotides are introduced into the ZMW "chamber". A light pulse is produced during synthesis, which can be detected. This process is performed in parallel in tens of thousands of ZMWs.

Nanopore sequencing in turn is based on entirely different chemistry. Here, a protein nanopore is placed in a polymer membrane that is electrically resistant. By setting a voltage across this membrane an ionic current is passed through the nanopore. When an analyte passes through the pore or near its opening, this event produces a characteristic disruption

11 https://www.illumina.com/systems/sequencing-platforms/miseq/specifications.html
12 https://www.pacb.com
13 https://nanoporetech.com

in current. The measurement of this current makes it possible to identify the molecule in question.

In particular, ONT technology can be miniaturized for portable analyses. Most notably, the MinION® was already used on board of the International Space Station. An even smaller device, designed for the use with smartphones, is under development at the time of writing[14].

2.3 REPRESENTATION OF DNA SEQUENCING DATA

In general, DNA sequencing is based on the determination of the nucleotide sequences of DNA fragments. The initial representation of nucleotide sequences is method-dependent. For example, in nanopore sequencing, the initial representation of a nucleotide sequence is in the form of a time series of a change in ion current of a nanopore. In sequencing by synthesis, the initial representation is in the form of images of fluorescently labeled nucleotides. However, eventually, all representations can be converted into a set of strings consisting of the characters A, C, G, and T. This conversion process is referred to as base calling, which is performed by different—often proprietary—software.

Sequencing data is stored in a number of different file formats. The most common ones are the FASTA format, the FASTQ format [Coc+10], and the SAM/BAM format [Li+09; The21]. These file formats can be considered as de-facto standards for the storage of nucleotide and amino acid sequences (FASTA), reads (FASTQ), and alignments (SAM/BAM). However, they exhibit several caveats; this fact is among the main drivers of standardization activities that aim to better represent sequencing data.

To illustrate one of these caveats, consider the raw read data produced during human WGS, as outlined in Section 1.1: in the FASTQ format, each base is stored as an 8-bit ASCII character, accompanied by a quality score (indicating the confidence in that the base was called correctly), which is also stored as 8-bit ASCII character. The FASTQ format also includes a string identifying each read. Sequencing the human genome (which consists of roughly 3 billion bp) at a coverage of 200× would yield a FASTQ file with a size of about 1,285 GiB. By this example alone is it evident that even simple modifications (such as the use of a naïve 2-bit encoding for nucleotides) would already yield major improvements, let alone the possible improvements that sophisticated compression techniques could bring to the table. However, the FASTA, FASTQ, and SAM/BAM formats are still widely used.

14 https://nanoporetech.com/products/smidgion

We detail the FASTA format, the FASTQ format, and the SAM/BAM format in Sections 2.3.1, 2.3.2, and 2.3.3, respectively.

2.3.1 *The FASTA Format*

The FASTA format is a textual file format for the representation of nucleotide and amino acid sequences. Each sequence is identified by a description line, which is also often referred to as header line or identifier line. Modern bioinformatics programs expect that the description line starts with ">". Typically, the description line only contains a unique identifier for the sequence. However, it can also contain additional information such as FASTA sequence identifiers[15] as defined by the NCBI. A single sequence can be stored in a single line, or it can be broken up and occupy multiple lines. Each nucleotide or amino acid in a sequence is represented with a single ASCII character according to the IUPAC nucleotide and amino acid codes. We refer to the tuple of description line and sequence as FASTA record.

As an example, Figure 2.2 depicts the sequence of isoform 1 of human serum albumin[16] in the FASTA format, as retrieved from the UniProtKB/Swiss-Prot database[17]. The description line starts with ">", followed by a FASTA sequence identifier ("sp|P02768|ALBU_HUMAN") and additional supplementary information. The actual amino acid sequence is broken up and stored across multiple lines.

As another example, Figure 2.3 depicts the first 283 nucleotides of the sequence of the HBB[18] gene in the FASTA format, as retrieved from the NCBI GenBank database[19].

2.3.2 *The FASTQ Format*

Similar to the FASTA format, the FASTQ format is a textual file format. It is mainly used for the representation of reads, i.e., nucleotide sequences, including corresponding quality scores. It is the de-facto standard for the storage of reads.

Each read is represented by a single FASTQ record, which consists of four lines. The first line contains the read identifier. It starts with "@" and is similar to the FASTA description line. Typically, sequencing machine

15 `https://ncbi.github.io/cxx-toolkit`
16 Human serum albumin is a protein found in human blood.
17 `https://www.uniprot.org`
18 HBB is a protein which is a part of hemoglobin A, the most common form of hemoglobin in adult humans.
19 `https://www.ncbi.nlm.nih.gov`

```
>sp|P02768|ALBU_HUMAN Albumin OS=Homo sapiens OX=9606 GN=ALB PE=1 SV=2
MKWVTFISLLFLFSSAYSRGVFRRDAHKSEVAHRFKDLGEENFKALVLIAFAQYLQQCPF
EDHVKLVNEVTEFAKTCVADESAENCDKSLHTLFGDKLCTVATLRETYGEMADCCAKQEP
ERNECFLQHKDDNPNLPRLVRPEVDVMCTAFHDNEETFLKKYLYEIARRHPYFYAPELLF
FAKRYKAAFTECCQAADKAACLLPKLDELRDEGKASSAKQRLKCASLQKFGERAFKAWAV
ARLSQRFPKAEFAEVSKLVTDLTKVHTECCHGDLLECADDRADLAKYICENQDSISSKLK
ECCEKPLLEKSHCIAEVENDEMPADLPSLAADFVESKDVCKNYAEAKDVFLGMFLYEYAR
RHPDYSVVLLLRLAKTYETTLEKCCAAADPHECYAKVFDEFKPLVEEPQNLIKQNCELFE
QLGEYKFQNALLVRYTKKVPQVSTPTLVEVSRNLGKVGSKCCKHPEAKRMPCAEDYLSVV
LNQLCVLHEKTPVSDRVTKCCTESLVNRRPCFSALEVDETYVPKEFNAETFTFHADICTL
SEKERQIKKQTALVELVKHKPKATKEQLKAVMDDFAAFVEKCCKADDKETCFAEEGKKLV
AASQAALGL
```

Figure 2.2: Sequence of isoform 1 of human serum albumin in the FASTA format. The description line starts with ">", followed by a FASTA sequence identifier ("sp|P02768|ALBU_HUMAN") and additional supplementary information. The amino acid sequence is broken up and stored across lines 2–12.

```
>NG_059281.1 Homo sapiens hemoglobin subunit beta (HBB), RefSeqGene (
    ↪ LRG_1232) on chromosome 11
AACGAATGAGTAAATGAGTAAATGAAGGAATGATTATTCCTTGCTTTAGAACTTCTGGAATTAGAGGACA
ATATTAATAATACCATCGCACAGTGTTTCTTTGTTGTTAATGCTACAACATACAAAGAGGAAGCATGCAG
TAAACAACCGAACAGTTATTTCCTTTCTGATCATAGGAGTAATATTTTTTTCCTTGAGCACCATTTTTGC
CATAGGTAAAATTAGAAGGATTTTTAGAACTTTCTCAGTTGTATACATTTTTAAAAATCTGTATTATATG
```

Figure 2.3: The first 283 nucleotides of the sequence of the HBB gene in the FASTA format.

vendors generate read identifiers in a proprietary systematic way. The second line contains the nucleotide sequence, where each nucleotide is represented with a single ASCII character according to the IUPAC nucleotide codes. The third line starts with "+" and contains an optional description. Usually this line is left empty; it then only contains "+" as separator between the nucleotide sequence and the quality scores. The fourth line contains the quality scores. A quality score q is a value indicating the confidence in a base call. The first widely adopted quality scores were produced by a base calling program called Phred [EG98], which linked the base-calling error probability p_{error} logarithmically to the so-called Phred quality score q:

$$q = -10 \cdot \log_{10} p_{error}. \tag{2.1}$$

Phred quality scores are typically offset to be able to represent them with a single ASCII character. For example, in the Sanger quality score format,

```
1 @SRR001665.1 071112_SLXA-EAS1_s_4:1:1:672:654 length=70
2 GCTACGGAATAAAACCAGGAACAACAGACCCAGCACATTAACAACAAAGGGTAAAAGGCATCATGGCTTC
3 +SRR001665.1 071112_SLXA-EAS1_s_4:1:1:672:654 length=70
4 IIIIIIIIIIIIIIIIIIIIIIIIEII9IIIEIIIIIIIIIIIIII4IIIIIGIHIIIIIII=IIIIIII
5 @SRR001665.2 071112_SLXA-EAS1_s_4:1:1:657:649 length=70
6 GCAGAAAATGGGAGTGAAAATCTCCGATGAGCAGCTTGATGCGACGACGCACCTCGTTGTTACGCACTTC
7 +SRR001665.2 071112_SLXA-EAS1_s_4:1:1:657:649 length=70
8 IIIIIIIIIIIIIIIIIIIIIIIII8II=II;IIIIIIIIIIIIIIIIIIIIIIIIIIIIII5IIII
```

Figure 2.4: Two FASTQ records (each consisting of four lines), obtained from the sequencing of *Escherichia coli*, strain K-12, substrain MG1655.

the offset 33 is added; hence Phred quality scores between 0 and 93 would be represented by the ASCII characters "!" (33) to "~" (126). As a short example, a base-calling error probability of $p_{error} = 0.001$ would translate to the Phred quality score $q = -10 \cdot \log_{10} 0.001 = 30$, which in turn would be represented by $30 + 33 = 63$ (i.e., "?") in the Sanger quality score format.

As an example, Figure 2.4 shows two FASTQ records, obtained from the sequencing of *Escherichia coli*, strain K-12, substrain MG1655, as retrieved[20] from the SRA[21]. Here, the read identifiers (lines 1 and 5) start with the SRA run accession number, followed by additional information. Note that, here, the third line contains a copy of the read identifier (this is done by convention in SRA data).

2.3.3 *The SAM/BAM Format*

Similar to the FASTA and FASTQ formats, the SAM format is a textual file format used for the representation of alignments, i.e., reads that have been aligned to reference sequences and hence are accompanied by additional alignment information and possibly further metadata.

The SAM format is composed of an optional header section and an alignment section. Header lines start with "@", while alignment lines do not. Each alignment line consists of 11 mandatory fields for essential alignment information such as the mapping position, and of a variable number of optional fields for further metadata, such as information

20 To generate Figure 2.4, we retrieved the data with the SRA run accession number SRR001665 from the SRA. Subsequently, we used the tool fastq-dump (with the command "`fastq-dump -stdout -maxSpotId 2 SRR001665`") from the SRA Toolkit (`https://github.com/ncbi/sra-tools`) to extract the shown FASTQ records. Finally, for visualization purposes, we trimmed the nucleotide sequences and the quality scores to a length of 70.

21 `https://trace.ncbi.nlm.nih.gov`

```
1 Coor       12345678901234  5678901234567890123456789012345
2 ref        AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT
3 +r001/1          TTAGATAAAGGATA*CTG
4 +r002           aaaAGATAA*GGATA
5 +r003         gcctaAGCTAA
6 +r004                      ATAGCT..............TCAGC
7 -r003                             ttagctTAGGC
8 -r001/2                                        CAGCGGCAT
```

Figure 2.5: Reads aligned to a reference sequence. Six reads are aligned to a reference sequence `ref`. Uppercase characters indicate nucleotides that match to the reference sequence; lowercase characters indicate nucleotides that do not match to the reference sequence (and that are hence "clipped" from the alignments). Dots represent a gap (here separating the split alignment). Example originally presented in [The21].

specific to the alignment software that was used. Note that all fields are delimited by tab characters.

We explain the SAM format with an example. For more details we refer the reader to the Sequence Alignment/Map Format Specification [The21].

Figure 2.5 illustrates six reads that have been aligned to a reference sequence (`ref`). The reference sequence is given coordinates (`Coor`) from 1 to 45; their remainders of the integer division by 10 are shown in line 1. Note that the reads include nucleotides that seem to be inserted with respect to the reference sequence (between reference sequence coordinates 14 and 15). `r001/1` and `r001/2` represent a read pair, `r002` is a single read, `r003` is a chimeric[22] read, and `r004` represents a split alignment. Uppercase characters indicate nucleotides that match to the reference sequence; lowercase characters indicate nucleotides that do not match to the reference sequence (and that are hence "clipped" from the alignments). Dots represent a gap (here separating the split alignment).

Figure 2.6 shows the data from Figure 2.5 formatted in the SAM format. The first two lines (starting with "@") contain the header section. The subsequent lines 3 to 8 constitute the alignment section. In the alignment section, column 1 identifies the read. Columns 3 and 4 identify the reference sequence and the mapping position, respectively. Column 6 contains the so-called CIGAR string, which contains a specially encoded version of the edit operations that are necessary to transform the read into the portion of the reference sequence that it maps to. Column 10 contains the actual nucleotides. Finally, column 11 would contain the

22 A chimeric read cannot be represented as a linear alignment; it is represented as a series of linear alignments that do not have large overlap. For more information we refer the reader to [The21].

```
1 @HD VN:1.6 SO:coordinate
2 @SQ SN:ref LN:45
3 r001   99 ref  7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
4 r002    0 ref  9 30 3S6M1P1I4M * 0   0 AAAAGATAAGGATA    *
5 r003    0 ref  9 30 5S6M       * 0   0 GCCTAAGCTAA       *
6 r004    0 ref 16 30 6M14N5M    * 0   0 ATAGCTTCAGC       *
7 r003 2064 ref 29 17 6H5M       * 0   0 TAGGC             *
8 r001  147 ref 37 30 9M         = 7 -39 CAGCGGCAT         *
```

Figure 2.6: Reads aligned to a reference sequence, represented in the SAM format. This example additionally shows two header lines (starting with "@"). Note that, here, quality scores are omitted (indicated by "*" in column 11. Example originally presented in [The21].

associated quality scores, but these are omitted (indicated by "*") in this example. Further optional columns are assumed not to be present.

A BAM file is the binary equivalent of a SAM file. In it, the same data is stored in a compressed binary representation. The BAM format utilizes BGZF[23], which implements block compression on top of the gzip file format [Deu96]. Basically, a BGZF file—and hence a BAM file—is a series of concatenated BGZF blocks, where each block is itself compliant to the gzip file format specification. BGZF files support non-sequential access through the so-called BAM file index[24], which is essentially a collection of offsets into the BGZF file.

2.4 DATA COMPRESSION

Data compression is the science of representing data in a compact form. The term compression always refers to two processes: compression and decompression, where decompression is also referred to as reconstruction. In the compression process, input data $\mathcal{X}$ is transformed into a compressed data representation $\mathcal{X}_c$ that is more compact than $\mathcal{X}$, i.e., it requires fewer bits (when stored in a binary form). In the decompression process, the compressed data representation $\mathcal{X}_c$ is used to generate a reconstruction $\mathcal{Y}$ of the data $\mathcal{X}$. In lossless compression the reconstruction $\mathcal{Y}$ is identical to $\mathcal{X}$; in lossy compression the reconstruction $\mathcal{Y}$ is allowed to be different from $\mathcal{X}$.

23 BGZF is specified in the SAM/BAM specification [The21].

24 Commonly the BAM file index is stored in a separate file with the extension ".bai", and it is usually stored along the associated BAM file.

The amount of achieved compression is expressed by the compression ratio, which is defined as

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}. \tag{2.2}$$

Occasionally, compression ratios are written explicit ratios. For example, a compression ratio of 5 may be notated as 5:1; i.e., the compressed data is five times smaller than the uncompressed data.

The input data can be of any type. It could for example be image data, video data, audio data, or DNA sequencing data. Compact representation of such data is achieved by identifying structures in the data, and by exploiting the knowledge about these structures. In the first place, the goal of compression is to reduce, and optimally to eliminate, all redundant information that is contained in the input data.

Regarding redundancy reduction, compression can be divided into two stages: modeling and coding. In the first stage, redundant information is described in the form of a mathematical model. The data is then expressed in terms of a difference—the residual—between the data and the model. In the coding stage, the residual is actually compressed by representing every residual element with as few bits as possible. Redundancy reduction does not involve the loss of information. The methods used to generate appropriate models and to efficiently code the data are therefore referred to as lossless compression techniques.

Data may also contain information that is considered to be irrelevant. For example, in the case of audio data intended for consumption by humans, data pertaining to frequencies outside of the human-audible frequency range do not need to be maintained and can simply be dropped. In other cases, for example in the case of chrominance[25] subsampling in video coding, chrominance information is represented using a lower resolution. Visual quality, however, is maintained because the human vision system has a finer spatial sensitivity to luminance differences than chrominance differences. In general, the techniques for irrelevancy reduction can be summarized by the term quantization. In all cases, a trade-off must be made between the amount of distortion that can be accepted after reconstruction of the data, and the compression that can be achieved by dropping parts of the data. In contrast to redundancy reduction, irrelevancy reduction does involve the loss of information. Quantization techniques are therefore referred to as lossy compression techniques.

The performance of different compression methods can be analyzed across three different dimensions: i) effectiveness: the compression (and

25 In video systems, the chrominance signal conveys the color information. It is paired with the luminance signal, which conveys the brightness (or "black-and-white") information.

distortion, in the case of lossy techniques) achieved by the method; ii) efficiency: the computational complexity; and iii) functionality: the additional features associated to the method, e.g., non-sequential access, etc. Depending on the specific usage scenario, priority can be shifted toward specific dimensions. For example, in a scenario where for example sequencing data is archived for long periods of time, effectiveness will have priority over efficiency; and in a scenario where sequencing data is supposed to be streamed over the internet, efficiency and functionality will have priority over effectiveness (if, of course, sufficient bandwidth is available).

In what follows we first introduce mathematical preliminaries for compression in Section 2.4.1. The basics concepts of information theory are then presented in Section 2.4.2. Subsequently, we detail the redundancy reduction stages modeling and coding in Sections 2.4.3 and 2.4.4. Finally, we present basics about quantization, i.e., irrelevancy reduction, in Section 2.4.5.

2.4.1 *Mathematical Preliminaries*

We introduce a few mathematical preliminaries necessary for the understanding of the concepts presented in this dissertation. These preliminaries include probability (Section 2.4.1.1), random variables (Section 2.4.1.2), probability distributions (Section 2.4.1.3), the concepts of marginal and conditional probability (Sections 2.4.1.4 and 2.4.1.5, respectively), and independence (Section 2.4.1.6), as well as expectation, variance, and covariance (Section 2.4.1.7).

2.4.1.1 *Probability*

We introduce the probability $P = P(e)$ of some elementary event e, i.e., an event which contains only a single outcome in the sample space[26], in the sense of the axiomatic definition by Kolmogorov [Kol33]. $P(e)$ is a real number between 0 and 1, inclusive:

$$0 \leqslant P(e) \leqslant 1. \tag{2.3}$$

The probability of the entire sample space is equal to 1, i.e., the probabilities $P(e)$ are normalized:

$$\sum_{e} P(e) = 1. \tag{2.4}$$

26 The sample space is the set of all possible outcomes of an experiment in the context of probability theory.

Also, any two mutually exclusive elementary events e_1 and e_2 satisfy:

$$P(e_1 + e_2) = P(e_1) + P(e_2). \tag{2.5}$$

In what follows, we extend this basic concept of probability.

2.4.1.2 *Random Variables*

A variable that can take on different values randomly is called a random variable. We give random variables lowercase names and write them in roman type, e.g., x, to distinguish them from "ordinary" scalars (that are written in italics, e.g., x). To discriminate the random variable itself from the values, i.e., states, it can take on, we write those states in italics, as we do for scalars. For example, x_1 and x_2 might be two possible values that the random variable x can take on.

A random variable only describes possible states; it must be coupled with a probability distribution that specifies how likely the occurrence of each of the states is. Also, random variables may be discrete or continuous. A discrete random variable has a finite number of states[27]. A continuous random variable has an infinite number of states, where the states are associated to the real numbers.

2.4.1.3 *Probability Distributions*

A probability distribution is coupled to a random variable or to a set of random variables. It indicates how likely it is that the random variable (or set of random variables) is to take on each of its states.

A probability distribution over discrete random variables is called PMF. PMFs are denoted with an uppercase P. The probability that a random variable x takes on state x is denoted as $P(x = x)$. We sometimes shorten this to $P(x)$. Sometimes we need to further disambiguate the probability distribution that a random variable follows. When then use the $\sim$ notation: $x \sim P$.

Probability distributions can also act on many random variables simultaneously. Such probability distribution are called joint probability distributions. The probability that random variable x takes on state x and that random variable y takes on state y at the same time is denoted as $P(x = x, y = y)$.

A probability distribution over continuous random variables is called PDF. PDFs are denoted with a lowercase p. A PDF does not give the probability of specific state directly. It must be integrated to find the

27 Note that the states do not have to be integers; they can also be named states.

actual probability that x lies in a certain interval, e.g., $\int_{[a,b]} p(x)dx$ in the case of a PDF coupled to a single random variable.

As an example for a PDF we consider a uniform probability distribution on a real interval $[a, b]$ with $b > a$. This distribution is given by $u(x; a, b)$. The ";" notation means "parametrized by", i.e., x is the function argument, and a and b are parameters that define the function. To ensure that u is normalized we set $u(x; a, b) = 0 \; \forall \; x \notin [a, b]$ and $u(x; a, b) = \frac{1}{b-a} \; \forall \; x \in [a, b]$.

2.4.1.4 *Marginal Probability*

Given a joint probability distribution over a set of random variables it is possible to calculate the probability distribution over a subset of the random variables. Such a probability distribution over the subset is called marginal probability distribution.

In the case of a joint probability distribution over two discrete random variables x and y, $P(x, y)$, the probability distribution $P(x)$ over the single variable x can be computed as $\sum_y P(x, y = y)$.

For continuous random variables integration is used instead of summation: $p(x) = \int_y p(x, y)dy$.

2.4.1.5 *Conditional Probability*

Let us now consider that an event depends on another event. The conditional probability that $x = x$ given $y = y$ is denoted as $P(x \mid y)$, in the case of discrete random variables x and y. The joint probability $P(x, y)$ can be expressed using the conditional probability and vice versa:

$$P(x, y) = P(x \mid y) \cdot P(y) \Leftrightarrow P(x \mid y) = \frac{P(x, y)}{P(y)}. \tag{2.6}$$

The formulation for continuous random variables is analogous.

Note that we can infer Bayes' theorem directly from Equation 2.6:

$$P(x \mid y) = \frac{P(y \mid x) \cdot P(x)}{P(y)}. \tag{2.7}$$

2.4.1.6 *Independence*

Two discrete random variables x and y coupled to the probability distributions $P_x(x)$ and $P_y(x)$, respectively, are identically distributed if and only if

$$P_x(x) = P_y(x) \Leftrightarrow P_y(y) = P_x(y). \tag{2.8}$$

Also, the occurrence of events described by x can be independent (i.e., not related) to the occurrence of events described by y. In this case the joint probability distribution $P_{x,y}(x, y)$ of both x and y can be expressed as a product of two probability distributions, where one only involves x and one only involves y. More specifically, x and y are independent if and only if

$$P_{x,y}(x, y) = P_x(x) \cdot P_y(y). \tag{2.9}$$

The formulations for continuous random variables are analogous.

In general, a collection of random variables is independent and identically distributed if each random variable is coupled to the same probability distribution as the others, and if all random variables are mutually independent.

2.4.1.7 *Expectation, Variance, and Covariance*

The expectation of some function $f(x)$ with respect to a probability distribution $P(x)$ is the mean value that f takes on when x is drawn from the probability distribution. For discrete random variables the expectation is defined as

$$\mathbb{E}_{x\sim P}[f(x)] = \sum_x f(x)P(x). \tag{2.10}$$

For continuous random variables the expectation is computed with an integral:

$$\mathbb{E}_{x\sim p}[f(x)] = \int_x f(x)p(x)dx. \tag{2.11}$$

The expectation of the function $f(x) = x^n$ with $n \in \mathbb{N} \setminus 0$ is referred to as n-th moment

$$M_n = \mathbb{E}_{x\sim P}[x^n]. \tag{2.12}$$

The first moment M_1 is known as the linear mean m_x. With the linear mean we can define the n-th central moment

$$C_n = \mathbb{E}_{x\sim P}\left[(x - m_x)^n\right]. \tag{2.13}$$

The second central moment C_2 is known as variance $\mathrm{var}(x)$. Its positive square root is the standard deviation σ_x.

The covariance provides information about the extent to which two values are linearly related to each other, as well as about the scale:

$$\mathrm{cov}(x, y) = \mathbb{E}_{x\sim P_x, y\sim P_y}\left[(x - m_x) \cdot (y - m_y)\right]. \tag{2.14}$$

High absolute values of the covariance mean that the values change a lot and at the same time are far from their mean values. If the sign of the covariance is positive, both random variables tend to take on relatively

high values simultaneously. If the sign of the covariance is negative, then one random variable tends to take on a relatively high value at the times that the other takes on a relatively low value and vice versa. Other measures such as correlation—as measured for example by the PCC—normalize the contribution of each random variable to measure only how much the variables are related, rather than also being affected by their scales.

The PCC, expressing the strength of the linear relationship between x and y can be expressed using the covariance $\mathrm{cov}(x, y)$ and the standard deviations σ_x and σ_y:

$$\rho_{xy} = \frac{\mathrm{cov}(x, y)}{\sigma_x \cdot \sigma_y}. \tag{2.15}$$

If x and y are uncorrelated, then $\rho_{xy} = 0$. If $|\rho_{xy}| = 1$, then x and y are linearly dependent.

Note that correlation and independence (see Section 2.4.1.6) are related but distinct concepts. If two random variables x and y are independent, then they are uncorrelated, i.e., they exhibit zero covariance. Also, two variables x and y that have nonzero covariance are dependent. Independence, however, is a distinct property from covariance. For two variables to have a covariance of zero, there must be no linear dependence between them. Independence is a stronger requirement than zero covariance, because independence also excludes nonlinear relationships. Nevertheless, correlation is an indicator for a predictive relationship that can be exploited in practice, e.g., in the modeling stage of a compression technique.

2.4.2 *Information Theory*

Information theory is based on the observation that knowing that a likely event has occurred is less informative than knowing that an unlikely event has occurred. A quantification of information should have the following properties:

1. Likely events should have a low information content, and events that are certain to occur should have no information content at all. Less likely events should have a higher information content.

2. Independent events should have additive information content.

Satisfying property 1 could easily be achieved by defining the information content of an event x as $-P(x)$. We can use the logarithm (with

respect to an arbitrary base) to satisfy property 2. The self-information of an event x is hence defined as

$$I(x) = -\log P(x). \tag{2.16}$$

By using the base-2 logarithm, the unit of self-information is bit. Hence, one bit is the amount of information gained by observing an event of probability $\frac{1}{2}$. The use of other bases is of course possible[28]. For example, in machine learning the logarithm with base e is frequently used; in this case the unit of self-information is "nats".

Self-information deals only with a single event x. By computing the expectation of the self-information with respect to the entire probability distribution $P(x)$ we obtain the entropy

$$H(x) = \mathbb{E}_{x\sim P}[I(x)] = -\mathbb{E}_{x\sim P}[\log P(x)] = -\sum_x P(x)\log P(x). \tag{2.17}$$

The entropy gives the average information that is expected in an event x drawn from probability distribution $P(x)$. Distributions close to the Dirac delta distribution[29] have low entropy; distributions close to the uniform distribution have high entropy. We illustrate this by by showing the entropy of a binary random variable in Figure 2.7.

We can also compute the conditional entropy of a conditional probability distribution $P(x\,|\,y)$ of two discrete random variables[30] x and y by averaging the self-information $I(x\,|\,y) = -\log P(x\,|\,y)$:

$$H(x\,|\,y) = -\sum_x \sum_y P(x,y)\log P(x\,|\,y). \tag{2.18}$$

It can be shown (see [Mus02] for a proof) that the conditional entropy is always smaller or equal to the (unconditional) entropy,

$$H(x\,|\,y) \leqslant H(x), \tag{2.19}$$

where equality holds if and only if x and y are independent (see Section 2.4.1.6). This can be exploited during coding, such as higher-order arithmetic coding (see Section 2.4.4.4).

---

28 As all logarithms are proportional, e.g., $\log_2(x) = \frac{\log_e(x)}{\log_e(2)}$, it does not matter which logarithm is used.

29 The Dirac delta distribution is defined by

$$\delta(x) = \begin{cases} \infty, & \text{if } x = 0, \\ 0, & \text{otherwise,} \end{cases}$$

constrained by $\int_x \delta(x)\,dx = 1$.

30 The conditional entropy can be generalized to include conditional probability distributions involving conditioning on an arbitrary number of random variables.

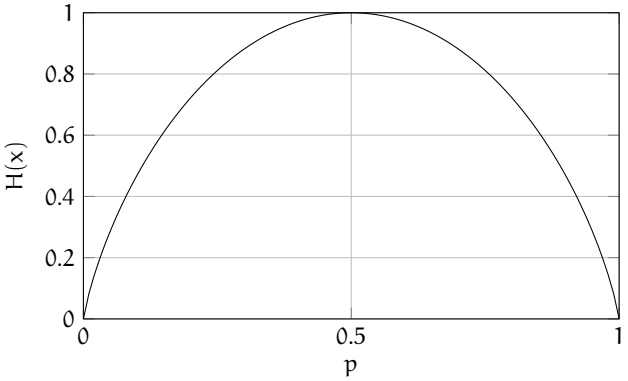


Figure 2.7: Entropy of a binary random variable. We show a plot of the entropy of the probability distribution $P(x = x)$ with $x \in \{0, 1\}$ and $P(x = 0) = 1 - p$ and $P(x = 1) = p$, respectively. The entropy is given by $H(x) = -\sum_x P(x) \log P(x) = -(1 - p) \cdot \log_2(1 - p) - p \cdot \log_2 p$ (using the base-2 logarithm). In the case that $p = 0.5$, i.e., $P(x)$ is a uniform probability distribution, the entropy is maximal. In the cases that $p = 0$ or $p = 1$, the entropy is minimal.

Analogously, we can also define the joint entropy[31]

$$H(x, y) = -\sum_x \sum_y P(x, y) \log P(x, y). \tag{2.20}$$

It can be shown (again, see [Muso2] for a proof) that the joint entropy is always smaller or equal to sum of individual entropies,

$$H(x, y) \leqslant H(x) + H(y), \tag{2.21}$$

where equality holds if and only if x and y are independent (see Section 2.4.1.6).

2.4.3 *Modeling*

In modeling, redundant information is described in the form of a mathematical model. The data is then expressed in terms of a difference—the residual—between the data and the model. In the coding stage, the residual is then actually compressed by representing every residual element

31 Again, we use two random variables for illustration. Of course, the joint entropy can also be generalized to joint probability distributions involving an arbitrary number of random variables.

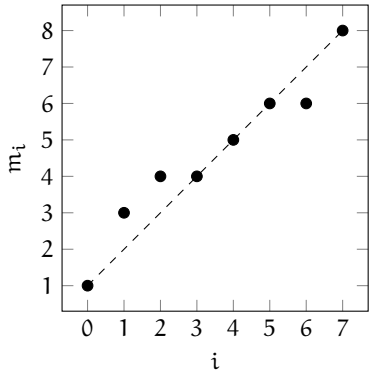


Figure 2.8: Plot of a vector of mapping positions. The data seems to fall on a straight line.

with as few bits as possible. Good models lead to more efficient compression algorithms. There are various strategies to creating mathematical models.

For example, consider the example of a set of reads that are aligned to a reference sequence. In this process a so-called mapping position is assigned to each read. Usually, reads and mapping position are subsequently sorted. Consider the following vector of mapping positions:

$$\mathbf{m} = (1\ 3\ 4\ 4\ 5\ 6\ 6\ 8)^{\mathsf{T}}. \tag{2.22}$$

The vector contains six distinct values; and using a very naïve binary code, each value could be represented by $\lceil \log_2 6 \rceil = 3$ bit.

However, by plotting $\mathbf{m}$ over its indices i we can explore the data and exploit its structure. The plotted data is shown in Figure 2.8. We see that the data seems to fall on a straight line. Therefore, a model for the data could be a straight line. Evaluating the straight line at the indices i gives the prediction of the model:

$$\hat{\mathbf{m}} = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)^{\mathsf{T}}. \tag{2.23}$$

We can now express the data in terms of a difference $\mathbf{d}$, i.e., residual, with respect to the model:

$$\mathbf{d} = \mathbf{m} - \hat{\mathbf{m}} = (0\ 1\ 1\ 0\ 0\ 0\ -1\ 0)^{\mathsf{T}}. \tag{2.24}$$

The residual vector $\mathbf{d}$ contains only three distinct values. Using the same naïve code, each value can now be represented using only $\lceil \log_2 3 \rceil = 2$ bit.

In general, mathematical models can be classified into physical models (detailed in Section 2.4.3.1) and probability models (detailed in Section 2.4.3.2). However, in many applications, it is not easy to use a single model to describe the data. In such situations, we can use a combination of several models—a composite model[32]—where only one model is active at any given time.

2.4.3.1 *Physical Models*

Knowledge about the physics of the data generation process can be used to construct an efficient model. For example, in Chapter 4 we devise a mathematical method that, in short, predicts the "importance" of quality scores that map to a specific locus. This "importance" is subsequently used to control the quantization of said quality scores. If the mathematical model always predicts the "importance" correctly, then it is possible to achieve an optimum in the equilibrium of rate and distortion (see Section 2.4.5). As can be seen from this example, models may not only serve the purpose of computing a residual; they can also serve as "proxy" models for controlling other parts of the compression process, such as quantization.

In many cases, however, the physics of data generation might be too complicated to be used to develop a model. If the physics of the problem is too complicated, we can build a model based on empirical observation of the statistics of the data.

2.4.3.2 *Probability Models*

Let us assume that a source generates symbols from an alphabet $\mathbb{A} = \{a_0, a_1, \ldots, a_{K-1}\}$ with cardinality $|\mathbb{A}| = K$. The symbols can be represented by a discrete random variable x that can take on the values in $\mathbb{A}$.

The simplest mathematical model is to assume that each symbol generated by the source is independent of every other symbol, and that each symbol occurs with the same probability. Following our assumption, we couple the alphabet with probabilities $P(a_k) = \frac{1}{K} \; \forall \; k \in \{0, 1, \ldots, K-1\}$.

The next level of complexity is to retain the assumption of independence, but remove the assumption of equal probability and assign each symbol its own probability of occurrence.

Given a probability model under the independence assumption we can compute the entropy of the source using Equation 2.17. It is also

32 The composite model is an exceptionally rich concept: there are myriad ways of controlling the model selection process; e.g., the model selection might be based on the statistical properties of already processed data.

possible to construct efficient codes that approach the entropy. However, these codes are only efficient if the independence assumption and the probability model are in accordance with reality.

In the case that the assumption of independence is not consistent with the observed data, it is generally possible to find better compression schemes by rejecting this assumption. One of the most popular techniques for representing dependencies in the data is through the use of Markov models[33] [Say06]. Let $\mathbf{x} = (x_0\ x_1 \cdots x_{N-1})^T$ be a symbol vector of length N, where each symbol is drawn from the alphabet $\mathbb{A}$. This vector is said to follow an m-th order Markov process if

$$P(x_{N-1} \mid x_{N-2}, \ldots, x_{N-m-1}) = P(x_{N-1} \mid x_{N-2}, \ldots, x_0). \tag{2.25}$$

Of course, m must be constrained to $1 \leqslant m \leqslant N-1$. In other words, knowledge of the past m symbols is equivalent to the knowledge of all past symbols. The values taken on by $x_{N-2}, \ldots, x_{N-m-1}$ are called the states of the process. The sequence $x_{N-2}, \ldots, x_{N-m-1}$ is also known as context. Given the alphabet size K the number of states is hence K^m.

Equation 2.25 indicates the existence of dependence between symbols. It does however not describe the details of the dependence; describing those details is an inherent part of the process to devise an appropriate model that efficiently describes the data.

Markov models are limited in the sense that the number of states K^m grows exponentially with m. A workaround is to estimate the involved conditional probabilities (rather than to empirically compute them using past data), e.g., using a neural network [Goy+20].

2.4.4 *Coding*

The term coding—or entropy coding—summarizes methods for the compression of data by representing it with as few bits[34] as possible. In coding, codes must satisfy the requirement of unique decodability, i.e., that individual code words can be recognized unambiguously. For fixed-length codes this is a trivial requirement. For variable-length codes unique decodability can be achieved by the use of prefix codes[35]. Coding methods can be grouped into different classes.

Static coding methods, such as Elias gamma coding [Eli75], Fibonacci coding, and Golomb coding [Gol66] (including Rice coding [RP71]) pro-

33 Here we use a specific type of Markov model called a discrete time Markov process.

34 In general, coding includes the generation of codes that are made up of an arbitrary numbers of code symbols. However, here, we focus on binary codes which are constructed using two code symbols, usually 0 and 1. We refer the reader to [Mus02] for the respective generalized formulations involving an arbitrary number of code symbols.

35 In a prefix code, no code word is a prefix of any other code word.

vide fixed, i.e., static, mappings of input symbols to variable-length code words. The computation of such mappings is independent from the actual input symbols to be compressed. However, if the statistical properties of the source are known in advance, the use of static coding methods is perfectly fine and can even be optimal. For example, if it is known that the input symbols follow a geometric probability distribution, then it can be shown that Golomb coding provides an optimal prefix code in this case [GV75].

Non-static coding methods, such as Huffman coding [Huf52], also map input symbols to variable-length code words. However, here, the codes are computed using the statistical properties of the source.

Arithmetic coding methods in turn, such as arithmetic coding [WNC87], range coding, and CABAC [MSW03], encode an entire message into a single interval which is represented by two numbers.

Most recently, the ANS family of coding methods has shown notable performance: ANS methods achieve the compression of arithmetic coding at a better computational performance than arithmetic coding [Dud14]. Similar to arithmetic coding methods, ANS methods encode an entire message into a single number (not an interval). Both arithmetic coding and ANS can be seen as numeral systems, i.e., systems for expressing numbers. Whereas "ordinary" numeral systems are optimized for expressing numbers following a uniform distribution, both arithmetic coding and ANS optimize for the specific distributions of the source.

For more detailed descriptions of various coding methods we refer the reader to [Say06].

In what follows, we first revisit the source coding theorem in Section 2.4.4.1. Subsequently, in Section 2.4.4.2, we briefly elaborate on the construction of Huffman codes. Then, in Section 2.4.4.3, we describe arithmetic coding in detail because range coding, a specialization of arithmetic coding, is used in our contributions TSC (see Chapter 3) and CALQ (see Chapter 4). Finally, in Section 2.4.4.4, we revisit CABAC, as our contribution GABAC (see Chapter 5) is built around a CABAC "core".

2.4.4.1 *The Source Coding Theorem*

In coding we consider a source that emits symbols. A source in its simplest form can be modeled by a discrete random variable x that takes on states x, and that is coupled to a probability distribution $P(x)$. In the context of coding we denote the set of states as alphabet $\mathbb{A}$. The symbols are the actual states the random variable takes on.

Suppose a source emitting symbols x with entropy $H(x)$. Further suppose that every symbol x_k is assigned a binary code word $\mathbf{c}_k \in \{0,1\}^{n_k}$

from a (variable-length) prefix code, where n_k is the length of code word $\mathbf{c}_k$. The mean code word length[36] of the entire code can then be computed as

$$\bar{n} = \sum_k P(x_k) n_k. \tag{2.26}$$

It can be shown (we refer the reader to [Mus02] for a proof) that the mean code word length satisfies the inequality

$$H(x) \leqslant \bar{n} < H(x) + 1, \tag{2.27}$$

which is known as source coding theorem. The difference

$$r = \bar{n} - H(x) \tag{2.28}$$

is known as code redundancy. Among the group of variable-length prefix codes, those codes constructed using Huffman coding (see Section 2.4.4.2) yield the lowest code redundancy. However, here, the minimum code word length is one bit. Hence, symbols whose occurrence is overly probable (i.e., $P(x) > 0.5$) can not be coded efficiently, i.e., in such a way that r becomes zero. To circumvent this issue, a fixed number of subsequent symbols can be combined into "supersymbol" blocks of length L. Under the assumption that all symbols are independent, the joint entropy of a block can be computed as the sum of individual entropies $\sum_l H(x_l)$ (see Equation 2.21). The source coding theorem can now be rewritten as

$$\sum_l H(x_l) \leqslant L \cdot \bar{n} < \sum_l H(x_l) + 1. \tag{2.29}$$

Incorporating the assumption that all symbols are identically distributed leads to

$$L \cdot H(x_l) \leqslant L \cdot \bar{n} < L \cdot H(x_l) + 1, \tag{2.30}$$

which can finally be rewritten as

$$H(x_l) \leqslant \bar{n} < H(x_l) + \frac{1}{L}. \tag{2.31}$$

Hence, as the block size L approaches infinity, Huffman coding theoretically approaches the entropy limit, i.e., zero code redundancy. However, the blocking of arbitrarily large groups of symbols is impractical because the complexity of a Huffman code is linear in the number of possibilities to be encoded. This number grows exponentially with the block size, limiting the scope of blocking in practice.

Other coding methods, such as arithmetic coding or ANS methods do not produce codes for single symbols (or supersymbols); they rather encode an entire message. Hence, the codes produced by these methods always approach zero code redundancy.

36 Note that the code word lengths must satisfy Kraft's inequality $\sum_k 2^{-n_k} \leqslant 1$ for the entire code to be a prefix code.

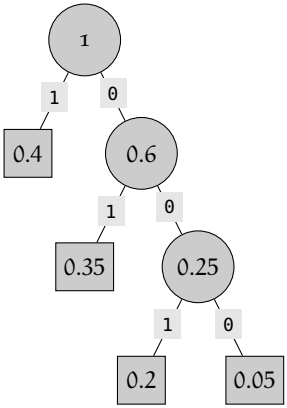


Figure 2.9: Construction of a Huffman code. Source symbol probabilities are shown in squared boxes. Probabilities of equivalent symbols are shown in circles.

2.4.4.2 *Huffman Coding*

As elaborated in Section 2.4.4.1, among the group of variable-length prefix codes, those codes constructed using Huffman coding [Huf52] yield the lowest code redundancy.

We briefly explain the construction of a Huffman code using the example shown in Figure 2.9. We assume a source generating symbols $x \in \{x_1, x_2, x_3, x_4\}$ with probabilities $P(x_1) = 0.4, P(x_2) = 0.35, P(x_3) = 0.2, P(x_4) = 0.05$. First, a binary tree is generated from the bottom up, taking the two least probable symbols and merging them into another equivalent symbol whose probability is equal to the sum of the two symbols. The process is repeated until there is only one symbol left (with probability 1). Second, the tree is read backwards, i.e., from top to bottom, assigning different bits to different branches[37]. The final Huffman code is shown in Table 2.2.

Note that the entropy of the source is approximately 1.74 bit per symbol, but that the mean code word length is 1.85 bit per symbol. The Huffman code still exhibits a code redundancy of around 0.11 bit per symbol. This is because the symbol probabilities are different from negative powers of two.

37 In our example, we assign ones to the "left" branches and zeros to the "right" branches. However, this assignment (left-1/right-0 or left-0/right-1) is arbitrary (as is the labeling of the branches using "left" and "right").

Table 2.2: Construction of a Huffman code. The corresponding binary tree is shown in Figure 2.9.

Symbol	Probability	Code Word
x_1	0.4	`1`
x_2	0.35	`01`
x_3	0.2	`001`
x_4	0.05	`000`

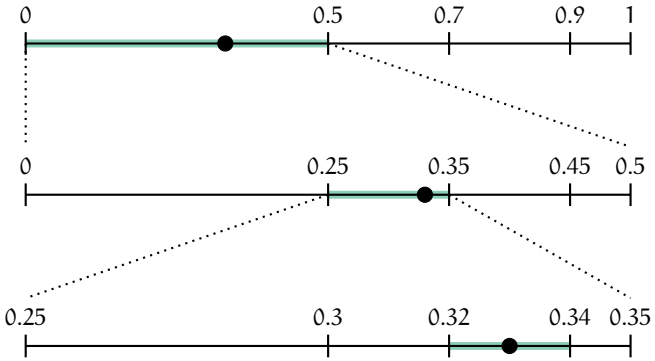


Figure 2.10: Arithmetic coding example. The decoding of the number 0.33 is shown.

2.4.4.3 *Arithmetic Coding*

In contrast to coding methods such as Huffman coding, where input symbols are mapped to variable-length codes, in arithmetic coding an entire message is encoded into a single number a with $0 \leqslant a < 1$. This number a lies in an interval, which is defined by two numbers. It is this interval that contains the message information, i.e., any number in it would represent the same message. In particular, it is only necessary to transmit enough digits (in whatever base) of a so that all numbers that begin with those digits fall into said interval. The key in arithmetic coding lies in that the interval can also be represented by repeated subdivision of the interval $[0, 1)$.

We briefly explain the working of arithmetic coding at the example decoding process shown in Figure 2.10, which shows said repeated subdivision process. For detailed descriptions of the encoding we refer the reader to [Muso2] and [Sayo6].

Consider the message $\mathbf{m} = (\mathrm{A\ B\ C})^{\mathsf{T}}$, and let us assume that $P(A) = 0.5$, $P(B) = 0.2$, $P(C) = 0.2$, and $P(D) = 0.1$. Also, let the message be encoded in the number 0.33. (Here we are using the decimal representation for clarity, instead of the binary representation. We also assume that there are only as many digits as needed to decode the message.)

The decoding begins with the same interval used by the encoder, $[0, 1)$. The decoding of the number 0.33 is now performed by repeated subdivision of this interval, according to the symbol probabilities. More specifically, the interval $[0, 1)$ is divided into the four sub-intervals $[0, P(A))$, $[P(A), P(A) + P(B))$, $[P(B), P(A) + P(B) + P(C))$, as well as $[P(A) + P(B) + P(C), 1)$. As can be seen at the top of Figure 2.10, the number 0.33 falls into the interval $[0, 0.5)$; hence, the first symbol must be A. Next, the interval $[0, 0.5)$ is divided into sub-intervals. Each sub-intervals constitutes a fraction of the current interval, which is proportional to the probability of the respective symbol in the current context[38]. The number 0.33 falls in the interval $[0.25, 0.35)$; hence, the second symbol must be B. After another interval subdivision (where again each sub-interval represents a fraction of the current interval proportional to the probability of that symbol in the current context) we find that 0.33 falls in the interval $[0.32, 0.34)$; hence we know that the third symbol must be C. Because the message length is known, decoding stops here. (In the case that the message length is not known, an additional "end-of-data symbol" must be used alternatively.)

One advantage of arithmetic coding over other compression methods is the possibility to employ adaptive probability models. This can be achieved by using the same initial probability model both at the encoder and at the decoder, and by adapting it equally and continuously on both sides.

The use of more sophisticated probability models is also possible. For example, arithmetic coding is often combined with Markov models (see Section 2.4.3.2). When using such higher-order modeling, current symbol probabilities are estimated using knowledge about preceding symbols. Here, the set of preceding symbols is referred to as the context.

2.4.4.4 *Context-Adaptive Binary Arithmetic Coding*

In arithmetic coding, in general, arbitrary interval subdivisions are used. Restricting the subdivisions to binary subdivisions is a method known as binary arithmetic coding. Using, in addition, an adaptive higher-order

38 The wording "in the current context" means that the probabilities might be adapted (depending on the current context) at each sub-interval division stage.

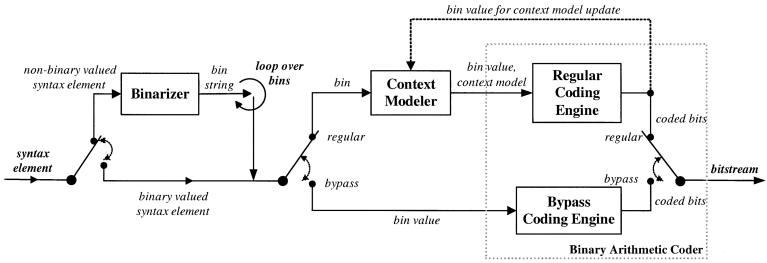


Figure 2.11: CABAC encoder block diagram (© 2003 IEEE [MSW03]).

probability model for the estimation of the current (binary) symbol probability is a method known as CABAC [MSW03].

CABAC is mostly known for its use in the AVC/H.264 [Wie+03] and HEVC/H.265 [Sul+12] standards. A block diagram of the CABAC encoder used in AVC/H.264 is shown in Figure 2.11.

The encoding process consists of at most three steps: binarization, context modeling, and binary arithmetic coding.

In the first step, non-binary symbols are binarized, i.e., they are uniquely mapped to binary sequences, so-called bin strings. For binary symbols, the binarization step is bypassed. The following steps are repeated for each bin.

In the so-called bypass coding mode, no context modeling is used, and every bin is encoded using binary arithmetic coding with a fixed probability model. Often a probability model is used that assumes equiprobability for all bins. The bypass coding mode allows for faster encoding and decoding, at the expense of compression efficiency.

In the so-called regular coding mode a context model provides the probability of each bin being 0 or 1. Context models can be selected from a set of available models, depending on the statistics of recently coded symbols. Binary arithmetic coding is used to encode each bin according to the selected context model. Finally, the selected context model is updated.

Regarding the use of binary arithmetic coding, in contrast to m-ary arithmetic coding, it is important to note that nothing is lost in terms of modeling, as the individual (non-binary) symbol probabilities can be recovered using the probabilities of the corresponding individual bins. For an illustration of this aspect we refer the reader to [MSW03].

The use of CABAC in video coding has been very successful because its structure has been heavily adapted to suit special video coding requirements, such as low complexity. However, in Chapter 5 we show that CABAC can also be applied to DNA sequencing data, providing

an important addition to the array of coding methods used in DNA sequencing data compression.

2.4.5 *Quantization*

Apart from redundancy, which is eliminated by the two-stage process of modeling and coding, data may also contain information that is considered irrelevant. The techniques for irrelevancy reduction can be summarized by the term quantization. For example, rounding a real number x to the nearest integer value constitutes a very simple type of quantizer: a uniform (mid-tread[39]) quantizer.

In general, quantization is the process of mapping input values from a large set to a smaller set. Hence, this process involves the loss of information. Quantization techniques are therefore referred to as lossy compression techniques. This loss can be quantified using a distortion function, which measures the cost of representing a symbol x by its associated quantized symbol $\check{x}$. An oft-used function is for example the squared-error distortion $d(x, \check{x}) = (x - \check{x})^2$.

In all cases, a trade-off must be made between the amount of distortion that can be accepted after reconstruction of the data, and the compression that can be achieved by dropping parts of the data. In general, coarser quantization, i.e., accepting a larger amount of distortion, leads to better compression, i.e., a lower rate. These matters are discussed in rate-distortion theory, which provides the theoretical foundations for lossy compression. Here, we briefly explain quantization using the example of a uniform quantizer; for a detailed review of rate-distortion theory we refer the reader to [Muso2] and [Sayo6].

Because the set of possible output values of a quantizer is countable, any quantizer can be decomposed into two distinct stages, which can be referred to as the quantization stage and the reconstruction stage, where the quantization stage maps the input value x to an integer quantization index k and the reconstruction stage maps the index k to the reconstruction value $\check{x}_k$. For the example uniform quantizer described above (rounding of a real number to the nearest integer), the quantization stage can be expressed as

$$k = \left\lfloor x + \frac{1}{2} \right\rfloor, \tag{2.32}$$

and the reconstruction stage can be expressed as

$$\check{x}_k = k. \tag{2.33}$$

39 Mid-tread quantizers have a zero-valued reconstruction level around the value 0, while mid-riser quantizers have a zero-valued classification threshold.

The decomposition into a quantization and a reconstruction stage illustrates how quantized data can be transferred over a communication channel: at the source the quantization stage is performed and the index information is then communicated to the sink, where the reconstruction stage is performed to produce an approximation of the original input data. In general, the quantization stage may use any function that maps the input data to the integer space of quantization indices, and the reconstruction stage can conceptually be a LUT mapping each quantization index to a corresponding reconstruction value.

3

COMPRESSION OF ALIGNED READS

As outlined in Chapter 2, Section 2.4, data compression techniques consist, in general, of the two stages modeling and entropy coding.

In the modeling stage, redundant information is described in the form of a mathematical model. Mathematical models can be grouped into two different types: physical models and probability models. In a physical model, knowledge about the physics of the data generation process is used to construct an efficient physical model (see Section 2.4.3.1). A more generic model type, however, is the group of probability models (see Section 2.4.3.2). These models use different levels of assumptions about the statistics of the data. However, probability models can always only be an approximation to the underlying physics of the data.

Given a mathematical model (of any type), next, the data is expressed in terms of a difference—the residual—between the data and the model. In the entropy coding stage, the residual is then actually compressed by representing every residual element with as few bits as possible.

In this chapter, we introduce a physical model to describe aligned reads. In contrast to the state of the art, our model is designed to exhibit a low memory footprint, and to be able to operate without external reference sequences. We couple our model with generic compression methods to evaluate its performance with respect to the state of the art.

This chapter is organized as follows. First, in Section 3.1, we review the state of the art, and we introduce our contribution TSC [VMO16]. In Section 3.2, we present the architecture of TSC in more detail. Here, we put special emphasis on the two main elements of TSC: the decoupling of read and alignment information into so-called descriptor streams, and the use of a "sliding window" to infer a so-called "local sliding consensus reference" for the prediction of reads. The sliding window and local sliding consensus reference elements have been adopted in the ISO/IEC 23092 series. Hence, in Section 3.3, we show how exactly these elements are integrated in ISO/IEC 23092-2:2020. To evaluate the performance of TSC with respect to the state of the art, we make use of a

previously published benchmark suite, that we co-authored. We detail our complete experiment setup in Section 3.4, and we present and discuss the results in Section 3.5. Finally, we conclude our findings in Section 3.6.

3.1 STATE OF THE ART AND CONTRIBUTION

Methods for the compression of nucleotide sequences can be grouped into three categories: genome compression methods, read compression methods, and methods for the compression of aligned reads. We introduce each of these categories in the following sections: genome compression is introduced in Section 3.1.1, read compression is outlined in Section 3.1.2, and the compression of aligned reads is presented in Section 3.1.3. Our contribution, TSC, belongs to the group of compression methods for aligned reads; hence, we introduce it in the corresponding Section 3.1.3.

3.1.1 *Genome Compression*

The first DNA-related compression tools aimed at the compression of entire genomes. In the years up to 2007 the field of genome compression was limited to compression tools that make use of dictionary-based methods. These early compression tools were specifically tailored to the properties (such as approximate repeats and palindromes) of genomes. Compression tools developed in this time span until 2007 include biocompress-2[1], DNACompress [Che+02], and XM [Duc+07].

Starting in 2009, tools were developed that encode genomes with the help of a reference genome, such as DNAzip [Chr+09], GRS [WZ11], GReEn [PPG11], ERGC[2] [SR15], and iDoComp [OHW15]. With these reference-based tools it is possible to compress a single human genome down to a few megabytes[3]. They are also particularly well-suited for the compression of genome collections from the same species. This is for example implemented in GDC [DG11b] and GDC 2 [DDN15].

3.1.2 *Read Compression*

The compression of reads differs significantly from the compression of entire genomes, since a set of reads exhibits intrinsic redundancy, due to

1 https://hal.inria.fr/inria-00180949

2 The authors of GDC and iDoComp published a comment (see [Deo+15]) on ERGC. They show that ERGC is, in general, outperformed by both GDC and iDoComp. This is in contrast to what the ERGC authors report in their paper. ERGC wins only in the (very) special case when one of the genomes (referential or target) contains mixed-cased letters.

3 For example, using iDoComp, the human genome assembly GRCh37, also known as hg19, can be compressed down to almost only 1 MB.

the coverage. In the literature, read compression is often referred to as "FASTQ compression", since the FASTQ format is the de-facto standard for the representation of reads. All read compression tools that can be found in the literature use FASTQ files as their encoder input, and their decoders analogously output FASTQ files. Tools for the compression of reads can be grouped into "classic" tools, that retain the ordering of reads, "reordering" tools, that reorder reads with the goal of achieving better compression, and "alignment" tools, that employ fast (approximate) alignment techniques as preprocessing before the actual compression. While read compression tools are focused on the efficient compression of the actual reads, i.e., nucleotide sequences, all of them also compress the read identifiers and the quality scores; however, for the read identifiers and the quality scores mostly generic compressors such as gzip are used.

The group of classic tools includes DSRC [DG11a], DSRC 2 [RD14], FQC [Dut+15], LFQC [NPR15], as well as Fqzcomp and Fastqz [BM13]. For an exhaustive review of these tools we refer the reader to [Num+16].

The group of reordering tools includes SCALCE [Hac+12], BEETL [Cox+12], ORCOM [GDR15], Mince [PK15], and FaStore [Rog+18], as well as HARC [CTW18] and SPRING[4] [Cha+19]. It is important to point out that changing the order of reads is a legitimate approach, since they constitute a set, as they are, in principle, randomly sampled fragments from the underlying DNA molecules. Again, for an extensive review of these tools we refer the reader to [Num+16].

Alignment tools include Quip [Jon+12], LEON [Ben+15], Kpath [KP15], LW-FQZip [Zha+15], and KIC [Zha+16]. Again, for a review of these tools we refer the reader to [Num+16], and to the associated literature.

3.1.3 *Compression of Aligned Reads*

As mentioned in Section 3.1.2, the compression of reads differs significantly from the compression of entire genomes. The same is true for the compression of aligned reads: techniques for their compression differ again significantly from the compression of entire genomes, and also from the compression of (unaligned) reads. Here, the redundancy introduced by the coverage is already easily accessible for compression algorithms, thanks to the alignment information. Moreover, aligned reads are typically also sorted by their mapping positions, which additionally eases the accessibility of the redundant information. In the literature, the compression of aligned reads is often referred to as "SAM compression", since the SAM format is the de-facto standard for the representation of

4 Regarding the compression of actual reads, i.e., nucleotide sequences, SPRING is based on HARC, with significant improvements and added support for variable-length reads.

aligned reads. All tools for the compression of aligned reads that can be found in the literature use SAM[5] files as their encoder input, and their decoders analogously output SAM files.

To address the compression of alignments, several specialized compression tools have been proposed in the literature. Similar to the read compression tools—that focus on efficient read, i.e., nucleotide sequence, compression, but also encode the read identifiers and quality scores—the tools for the compression of aligned reads are focused on the compression of reads and their alignment information. However, most of these tools also include capabilities for the compression of read identifiers and quality scores. Again, here, mostly generic compressors such as gzip are used. These tools for the compression of alignments can be classified into reference-based tools and variation-sensitive tools.

Reference-based tools mainly include those tools that implement a version of the CRAM format specification[6]. CRAMTools[7] is an implementation of the CRAM 2.0 format. It decouples the different SAM fields into separate streams and applies a variety of compression techniques to each stream. For example, mapping positions are encoded using a combination of delta coding and Golomb coding [Gol66]. The actual read information is encoded in a reference-based manner. Scramble [Bon14], in turn, is an implementation of the CRAM 3.0 format, focused on an optimized implementation in the C language. Scramble estimates, already during encoding, the best compression technique for each stream, based on the corresponding statistics. Unlike CRAMTools, Scramble can also operate in a non-reference-based mode, where the reference sequence is re-generated from the alignment data and stored alongside the other compressed information.

The only variation-sensitive tool that can be found in the literature is DeeZ [HNS14], which is primarily a reference-based tool. However, additionally, it performs a local assembly of the reads, with the goal of modifying the corresponding reference sequence, such that it matches the actual read set to be compressed. This way, common small variations need to be encoded only once.

The state of the art in compression of aligned reads is primarily based on the availability of external reference sequences. (Note that Scramble can also operate in a non-reference-based mode.) In contrast to the state of the art, our contribution, TSC [VMO16], is designed with the goals of achieving a low memory footprint and operating without external

5 Most compression tools use HTSlib (`https://www.htslib.org`) for accessing SAM files. Hence, most of them are also able to accept BAM files as input (and to output BAM files).

6 The most recent version, specifying the CRAM 3.0 format, is available at `https://samtools.github.io/hts-specs/CRAMv3.pdf`.

7 `https://github.com/enasequence/cramtools`

reference sequences. TSC utilizes a sliding window (i.e., a permanently updated short-time memory) to track recent reads. The information residing in the sliding window is then used to infer a so-called "local sliding consensus reference", i.e., an implicitly assembled local part of the underlying DNA sequence, for the prediction of reads. Because the local sliding consensus reference is directly generated from the subject reads, TSC can be classified, along with DeeZ, as a variation-sensitive tool.

3.2 TSC ARCHITECTURE

TSC is built around two main elements: the decoupling of read and alignment information into so-called descriptor streams, and the use of a sliding window to infer a so-called local sliding consensus reference for the prediction of reads. We start explaining these elements by describing the TSC architecture from the encoder point of view.

As input, TSC requires reads, i.e., nucleotide sequences, and their alignment information, sorted by mapping position. Similar to the tools for the compression of aligned reads that can be found in the literature, TSC uses files in the SAM format as encoder input and decoder output. In the SAM format (see Section 2.3.3), the alignment information for a specific read, i.e., nucleotide sequence (referred to with the mnemonic "SEQ"), is composed by the corresponding reference sequence identifier ("RNAME"), the mapping position ("POS"), and the so-called CIGAR string ("CIGAR"), which contains a specially encoded version of the edit operations that are necessary to transform the read into the portion of the reference sequence that it maps to. Note that the TSC encoding algorithm does not require the reference sequence itself.

In the first step, the TSC encoder transforms each read, along with its alignment information, into an equivalent representation. This equivalent representation consists of a set of "descriptors" that convey the same information that is present in the RNAME, POS, CIGAR, and SEQ columns of the SAM format, but in a form that allows for more efficient compression.

In TSC, there are two different equivalent representation types: so-called "I-records" and "P-records". In analogy to I-frames in video coding, the I-records serve as starting points for sequences of predicted P-records. Hence, I-records are used for the first alignment in a block, and also in a few special cases, where the prediction is unfeasible (e.g., because there does not exist any overlap between the current read and the local sliding consensus reference) or not beneficial (e.g., because there are too many modifications in the current read with respect to the local

sliding consensus reference). In TSC, blocks have a defined maximum size, $n_{b,max}$. Upon reaching the maximum size, a new I-record is inserted.

An I-record consists of the following set of descriptors (we use mnemonics, such as "exs" for "expanded nucleotide sequence" to be able to efficiently refer to specific descriptors): `rname`, `pos`, `cigar`, and `exs`. These descriptors exhibit the following semantics:

- `rname`: This descriptor contains the reference sequence identifier. It contains the same information that is present in the RNAME column of the SAM format.
- `pos`: This descriptor contains the mapping position. It contains the same information that is present in the POS column of the SAM format.
- `cigar`: This descriptor contains the CIGAR string. It contains the same information that is present in the CIGAR column of the SAM format.
- `exs`: This descriptor contains the so-called "expanded" nucleotide sequence. The expanded nucleotide sequence is generated from the original nucleotide sequence (SAM column "SEQ") by removing from it nucleotides that are inserted with respect to the reference sequence; and padding it with an auxiliary symbol[8] where it exhibits deletions with respect to the reference sequence. (The information about insertions and deletions is available in the corresponding CIGAR string.)
- `inserts`: This descriptor contains the inserted nucleotides that have been removed from the original nucleotide sequence.

Once an I-record has been generated, its descriptors `pos` and `exs` are pushed to the sliding window, which can be thought of as fixed-size circular buffer. For each subsequent alignment, the two descriptors `pos` and `exs` are also pushed to the circular buffer. The circular buffer is configured to have a defined maximum capacity $n_{c,max}$, measured in number of I- and P-records. If this threshold is reached, oldest descriptors are removed from it to make room for new descriptors.

To encode subsequent alignments, a so-called "local sliding consensus reference" is built from the information in the circular buffer. Assume that the circular buffer operates at its maximum capacity. (In practice, for example, we use a maximum capacity of $n_{c,max} = 10$, i.e., a maximum of

8 In TSC the symbol "?" is used at those positions; however, the specific symbol is not of relevance.

ten `pos-exs` pairs is kept in the circular buffer.) Then, for each position, a consensus nucleotide is derived by majority vote[9] from the expanded nucleotide sequences in the circular buffer. This yields the local sliding consensus reference which now can be used to encode subsequent reads.

The encoding of subsequent reads works as follows. First, the original nucleotide sequence is expanded again. Then, the expanded nucleotide sequence is encoded differentially with respect to the local sliding consensus reference. This encoding yields a P-record that consists of the following descriptors: `posoff`, `cigar`, `inserts`, `modcnt`, `modpos`, `modbases`, and `trail`. These descriptors exhibit the following semantics:

- `posoff`: This descriptor contains the offset of the mapping position of the current nucleotide sequence with respect to the most recent mapping position that resides in the circular buffer (recall that TSC assumes that the input data is sorted ascending by mapping position).
- `cigar`: This descriptor contains the current CIGAR string. It is the same descriptor as used for I-records.
- `inserts`: This descriptor contains the inserted nucleotides that have been removed from the original nucleotide sequence. It is the same descriptor as used for I-records.
- `modcnt`: The number of modified nucleotides with respect to the local sliding consensus reference.
- `modpos`: The positions of the modified nucleotides, relative to the current mapping position.
- `modbases`: The actual modified nucleotides.
- `trail`: Those nucleotide that are not covered by the local sliding consensus reference. Recall that TSC best operates on data that is sorted ascending by mapping position[10]. Hence, eventually, new reads will extend on the "right" with regard to prior reads. Eventually, the `trail` descriptor will contain something that looks similar to a reference sequence.

Finally, a special descriptor is needed to be able to distinguish between I-records and P-records (because the descriptors `cigar` and `inserts` are shared by both types of records). This is the `ctrl` descriptor, which signals

9 The original implementation of TSC used a special policy to select expanded nucleotide sequences from the sliding window; for the details we refer the reader to [VMO16].

10 Still, if the algorithm encounters a violation of this requirement, new I-records may be inserted.

either an I-record or a P-record. Note that, to handle inconsistencies in the input SAM file (which are produced by aligners), and to provide a generic fallback solution, TSC also implements so-called M-records, which are virtually a copy of the corresponding SAM fields RNAME, POS, CIGAR, and SEQ. In general, compared to the proportion of I- and P-records, the proportion of M-records is negligible. For example[11], in the case of item 16 (see Table 3.1) TSC generates 1,390 I-records (0.0105%), 13,174,289 P-records (99.9885%), and only 131 M-records (0.0010%).

Note that, as the TSC technology was submitted to the standardization process of the ISO/IEC 23092 series, the semantics of some TSC descriptors overlap with some descriptors specified in ISO/IEC 23092-2:2020. In particular, the TSC descriptors `modpos` and `modbases` correspond semantically to the ISO/IEC 23092-2:2020 descriptors "mmpos" and "mmtype".

Finally, each stream of descriptors is subject to entropy coding. Here, we make the assumption that statistical dependencies between the different descriptors are negligible. Hence, we encode each descriptor stream separately. Except for the numerical `posoff`, `modcnt`, and `modpos` streams, which are encoded using range coding, all descriptor streams are encoded using zlib [Deu96]. Here, the actual compression is based on the LZ77 algorithm [ZL77] and Huffman coding [Huf52].

In summary, we devised a physical model to describe aligned reads. This physical model consists of the sliding window, and of the local sliding consensus reference that is built from it. The residual—the difference between the data and the model—takes the form of the descriptor streams, which are finally subject to entropy coding.

3.3 INTEGRATION OF TSC IN MPEG-G

The TSC technology was submitted to the standardization process of the ISO/IEC 23092 series, and it is now integrated in its part 2 as "local assembly" reference computation algorithm. To illustrate this, we show and explain the two relevant decoding processes from ISO/IEC 23092-2:2020. The first decoding process ("process for adding a decoded aligned read to the list crBuf") specifies the sliding window mechanism. The second decoding process ("process for the construction of the reference") specifies the building of the local sliding consensus reference from the data maintained in the sliding window.

Figure 3.1 shows the first decoding process, the "process for adding a decoded aligned read to the list crBuf", as specified in Clause 11.3.5.2 of ISO/IEC 23092-2:2020. This process specifies the sliding window mecha-

11 As in Section 3.4, we use the empirically derived parameters $n_{b,max} = 10000$ and $n_{c,max} = 10$ (see [VMO16]).

> The inputs to this process is an array crBuf[][] which contains crBufNumReads reads of size in bytes equal to crBufSize.The output of this process is the updated array crBuf[][] and the updated variables crBufNumReads and crBufSize.
>
> This process consists of the following steps:
>
> 1. If the variable crBufSize plus the length in bases of the already decoded aligned read is greater than cr_buf_max_size, the oldest reads are removed from the array crBuf[][] until crBufSize plus the size of the already decoded aligned read is smaller than or equal to cr_buf_max_size.
> 2. The last decoded read is added to the array crBuf[][] as newest read.

Figure 3.1: ISO/IEC 23092-2:2020, Clause 11.3.5.2, "Process for adding a decoded aligned read to the list crBuf".

nism. The contents of the sliding window are stored in the array named "crBuf" (which stands for "computed reference buffer"). More specifically, the array crBuf contains multiple decoded nucleotide sequences. The number of nucleotide sequences maintained in the array crBuf is stored in the variable named "crBufNumReads". The size in bytes of the array crBuf is recorded in the variable named "crBufSize". Old data is ejected from the array crBuf based on it reaching a maximal size in bytes, given by the encoding parameter cr_buf_max_size. This encoding parameter is equivalent to the TSC parameter $n_{c,max}$ (see Section 3.2). However, note that, where the TSC parameter $n_{c,max}$ specifies a maximum circular buffer capacity in a number of nucleotide sequences, the encoding parameter cr_buf_max_size is given in bytes. By giving the size in bytes, an encoder can better control the memory usage of the decoder, since read lengths might be highly variable. This mechanism enables memory-efficient encoding and decoding, as only the array crBuf needs to be kept in memory (in comparison to keeping an entire reference sequence in memory). What is more, as the maximal size of the array crBuf is defined by an encoding parameter, the memory consumption of decoding can already be defined during encoding.

Figure 3.2 shows the second decoding process, the "process for the construction of the reference", as specified in Clause 11.3.5.3 of ISO/IEC 23092-2:2020. (Note that, here, dots indicate additional process parts which are present in ISO/IEC 23092-2:2020, but which are not repeated here for brevity.) Here, we need to point out that, to execute this second decoding process, the position on the reference sequence of each nu-

> The input to this process is an array crBuf[][] containing at least one aligned read and the position on the reference sequence of each nucleotide.
>
> The output of this process is an array refBuf[] containing a sequence of consensus symbols.
>
> For each position covered by aligned reads in the array crBuf[][], the consensus symbol is derived as follows:
>
> 1. Collect all bases mapping to the current position.
> 2. Count the occurrences of each symbol.
> 3. If two symbols s_i, s_j (...) have the same maximum number of occurrences, then select s_i as consensus symbol.
> 4. Otherwise, select the symbol with the maximum number of occurrences as consensus symbol.
> 5. Append the consensus symbol to the array refBuf[].
> 6. ...
>
> The result of the decoding process described above is a reference sequence contained in the array refBuf[] ...

Figure 3.2: ISO/IEC 23092-2:2020, Clause 11.3.5.3, "Process for the construction of the reference". Dots indicate additional process parts which are present in ISO/IEC 23092-2:2020, but which are not repeated here for brevity.

cleotide in the array crBuf is required. This process specifies the building of the local sliding consensus reference from the data which is maintained in the array crBuf, i.e., the sliding window. Via majority vote, a consensus nucleotide is derived per position covered by nucleotide sequences stored in the array crBuf. The result of this decoding process is an array named "refBuf"—the incarnation of the local sliding consensus reference—which is subsequently used to decode the next ISO/IEC 23092-2 record. Note that the array refBuf is re-computed for each decoded nucleotide sequence. Hence, this scheme introduces an additional computational overhead, compared to the case where a reference sequence can be held available in memory. However, this overhead can be controlled by the encoding parameter cr_buf_max_size.

3.4 EXPERIMENT SETUP

We evaluate the performance of TSC with respect to the state of the art. We use the empirically derived parameters $n_{b,max} = 10000$ and $n_{c,max} = 10$

for TSC (see [VMO16]). For this purpose, we used the benchmark suite[12] published by Numanagić et al. [Num+16]. Because TSC is implemented as a single-threaded software, we report only the single-threaded performance of all tools here. (The benchmark suite is capable of performing simulations involving different numbers of threads.) All experiments were performed on a computer equipped with an Intel Core i9-9900K CPU and 64 GiB of RAM, running openSUSE Leap 15.2.

The state of the art consists of the general-purpose compressors gzip[13], the reference-based alignment compressor Scramble (version 1.14.6), implementing the CRAM 3.0 format, and the variation-sensitive alignment compressor DeeZ (version v1.9-beta1). Although Scramble has been designed as a reference-based tool, it can also be operated in a non-reference-based mode.

The compressed output of each of the tools can be split unambiguously into information pertaining to: i) nucleotide sequences and mapping information; ii) quality scores; iii) read identifiers; and iv) auxiliary fields. Here, we are only interested in the information pertaining to the nucleotide sequences and mapping information, which is associated to the SAM fields RNAME, POS, CIGAR, and SEQ; and these are the results that we report in Section 3.5. We refer the reader to [Num+16] for an exhaustive evaluation that includes the other categories.

We carefully selected representative test data from the MPEG-G Genomic Information Database [ISO20]. This database can be regarded as statistically meaningful, since it contains an abundance of sequencing data exhibiting a wide array of different characteristics. In particular, the database includes sequencing data that was generated in different experiment settings: it contains WGS data, metagenomic sequencing data, and RNA sequencing data. Also, the database contains data that originates from different species: *Drosophila melanogaster*, *Homo sapiens*, *Theobroma cacao*, *Saccharomyces cerevisiae*, *Escherichia coli*, *Pseudomonas aeruginosa*, and the virus ΦX174. Finally, the data was generated with various sequencing technologies such as sequencing by synthesis, SMRT sequencing, nanopore sequencing, and ion semiconductor sequencing.

Because of the sheer size of the database, we selected a subset of the data for our simulations, similar to [Num+16]. Notably, we only selected WGS data, to be able to draw conclusions with respect to the use of reference sequences. We also paid attention to select data that is different in the other two data dimensions (species and sequencing

12 https://github.com/sfu-compbio/compression-benchmark

13 The benchmark suite actually uses pigz (`https://zlib.net/pigz`), version 2.3.3. However, here, we are only evaluating the single-threaded performance, which lets pigz be equivalent to gzip.

Table 3.1: Selected test data from the MPEG-G Genomic Information Database [ISO20]. Particular attention was paid to species and sequencing technology diversity in the selection of test data.

Item	Species	Sequencing Technology	Coverage
3	*H. sapiens*	SMRT sequencing	15×
5	*H. sapiens*	Sequencing by synthesis	2×
9	*H. sapiens*	Ion semiconductor seq.	0.6×
16	*E. coli*	Sequencing by synthesis	420×
19	*D. melanogaster*	SMRT sequencing	75×
23	*H. sapiens*	Sequencing by synthesis	30×

technology). Specifically, we selected items 3, 5, 9, 16, 19, and 23. These items contain WGS data pertaining to the species *Homo sapiens* (items 3, 5, 9, 23), *Escherichia coli* (item 16), and *Drosophila melanogaster* (item 19). Also, the data were produced using different sequencing technologies: items 3 and 19 were produced using SMRT sequencing, items 5, 16, and 23 were produced using sequencing by synthesis, and item 9 was produced using ion semiconductor sequencing.

What is more, each item exhibits some special characteristics, making the entire set of items a good stress test for the selected set of codecs. Item 3 is from the widely used NA12878 (*Homo sapiens*) sample, but this time sequenced using SMRT sequencing technology. Item 5 exhibits a low and extremely uniform coverage of 2×. Item 9 was produced using ion semiconductor sequencing technology, which leads to large variations in the sequencing depth (the major factor leading to the extremely low coverage of 0.6×). Item 16 contains high-coverage data (420×) of a small genome (*Escherichia coli*). Item 19 contains uncorrected variable-length long reads with complex and lengthy CIGAR strings. Finally, item 23 exhibits a much higher degree of sequence variation than usual. Table 3.1 summarizes the details of the selected data.

3.5 RESULTS AND DISCUSSION

Table 3.2 shows the compressed sizes that were achieved by all codecs on all test items. As mentioned in Section 3.4, the compressed output of each tool can be split unambiguously into information pertaining to, on the one hand, reads and mapping information, as well as all remaining information, such as quality scores, on the other hand. Here, we only show the information pertaining to the reads, i.e., nucleotide sequences, and mapping information. This is the information that is associated to the

Table 3.2: Compressed sizes in MiB achieved by all codecs on all test items. The compressed data contains the read, i.e., nucleotide sequence, and alignment information (i.e., the information corresponding to the SAM fields RNAME, POS, CIGAR, and SEQ). At the bottom the compressed sizes (gzip) of the corresponding reference sequences are shown.

Codec \ Item	3	5	9	16	19	23
DeeZ	6,964	137	32	19	1,502	3,921
gzip	14,564	1,047	61	54	4,623	8,226
Samtools (BAM)	14,822	1,142	75	51	4,621	8,646
Scramble (CRAM 3.0)	6,440	129	36	17	1,303	3,882
—w/o ref.	14,852	929	92	48	4,747	7,235
TSC	14,522	1,079	43	24	2,288	8,205
Reference	905	905	905	1.4	43	905

SAM fields RNAME, POS, CIGAR, and SEQ. In the last row of Table 3.2 we also report the sizes of the corresponding reference sequences, as obtained by compressing them with gzip. The reference sequence sizes have to be taken into account when comparing the reference-based tools, DeeZ and Scramble, to the non-reference-based tools gzip, Samtools, Scramble (without reference), and TSC. Note that Samtools implements the BAM format, and that Scramble implements the CRAM 3.0 format.

To be able to better interpret the data presented in Table 3.2, we first normalized all compressed sizes with respect to the associated BAM size. We use the BAM format as anchor because in the case of aligned read compression we regard the BAM format as the de-facto standard. Second, we plotted the compressed sizes in the form of a bar chart, where bars are grouped by item. For the DeeZ and Scramble bars we stack the corresponding compressed reference sizes on top, because the compressed sizes of the non-reference-based tools already include some kind of compressed representation of an associated reference sequence. This visualization is shown in Figure 3.3. We discuss the results shown in Figure 3.3 separately per item.

In the case of item 3, gzip, "Scramble w/o ref.", and TSC achieve a compressed size similar to the BAM size. For gzip this is expected: recall that the BAM format utilizes BGZF, which implements block compression on top of the gzip file format [Deu96] (see Section 2.3.3). "Scramble w/o ref." and TSC both build some kind of reference sequence from the available read data. Hence, in general, they are expected to exhibit similar compression performances. However, here, in the case of item 3,

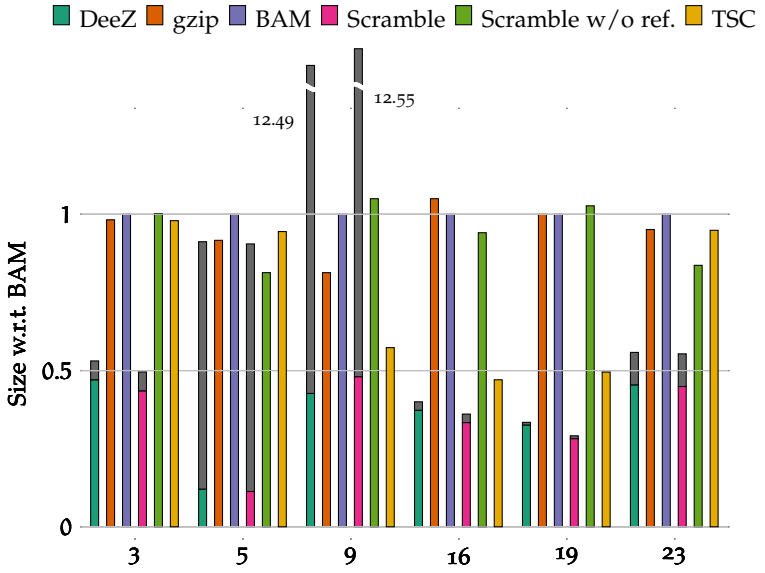


Figure 3.3: Compressed sized, with respect to BAM, achieved by all codecs on all test items. For the DeeZ and Scramble bars we stack the corresponding compressed reference sizes on top, because the compressed sizes of the non-reference-based tools already include some kind of compressed representation of an associated reference sequence.

"Scramble w/o ref." and TSC are in this case clearly outperformed by DeeZ and Scramble (with reference), even taking into account the sizes of the compressed references that are added to the DeeZ and Scramble (with reference) results. We suspect that this behavior has its roots in the sequencing technology that was used to generate item 3: SMRT sequencing technology. Reads produced using SMRT sequencing technology exhibit, in general, high amounts of insertions. The coding methods implemented in DeeZ and Scramble (with reference) seem to be better suited for this task.

In the case of items 16, 19, and 23, compared to item 3, we observe qualitatively similar results. However, for items 16 and 19, TSC clearly outperforms "Scramble w/o ref.", joining the group of the reference-based tools. We can conclude that TSC better exploits the redundancy present in the short and low-error reads of item 16, compared to "Scramble w/o ref.". Regarding item 19, things lie differently. Item 19 is again produced using SMRT sequencing technology. Here, in contrast to item 3, the coverage is 75× (item 3: 15×). Recall that in our experiments we con-

figured TSC to use a sliding window size of $n_{c,max} = 10$. This means that, in principle, TSC should exhibit a similar performance on items 3 and 19. However, the coverage is always only an average over the sequencing depth at all loci. Hence, we can conclude that the sequencing depth for both items (3 and 19) is slightly variable. In the case of item 19, however, the coverage is throughout high enough for TSC to build efficient local sliding consensus references.

Items 5 and 9 are two special cases. Item 5 exhibits a low and extremely uniform coverage of $2\times$. Hence, one would expect that reference-based and non-reference-based codecs achieve similar compressed sizes, and this is what we can observe from Figure 3.3. Item 9 contains data with extremely large variations in the sequencing depth, leading to the extremely low coverage of $0.6\times$. Here, taking into account the compressed sizes of the associate reference sequences, we clearly see that the reference-based tools are outperformed by the non-reference-based tools. In particular, TSC is the best-performing codec.

TSC was designed to operate with an extremely low memory footprint. (Recall that in our experiments we use a sliding window size of only $n_{c,max} = 10$.) Hence, we also evaluate the memory usage of each codec. Table 3.3 shows, for all codecs, the achieved compressed sizes with respect to BAM, averaged over all items, as well as the maximum RSS[14], additionally averaged over encoding and decoding. From the table we can see that the generic codecs gzip and Samtools have the lowest average memory usage (10 MiB and 11 MiB, respectively). The reference-based codecs operate with significantly higher memory usage: DeeZ requires a maximum RSS of 6,774 MiB on average, and Scramble requires 1,166 MiB on average. The non-reference-based mode of Scramble requires only 783 MiB on average. The lowest memory usage among the specialized codecs is achieved by TSC, with only 563 MiB on average. Additionally, compared to the non-reference-based mode of Scramble, TSC reaches an average compressed size of 0.8305 with respect to BAM, where Scramble (without reference) only reaches an average compressed size of 0.9976, barely outperforming the anchor BAM.

3.6 CONCLUSION

We introduced a physical model, TSC, to describe aligned reads. In contrast to the state of the art, our model is designed to exhibit a low memory footprint, and to be able to operate without external reference sequences.

14 The resident set size (RSS) is the amount of memory occupied by a process that is in main memory (RAM). The rest of the occupied memory is in the swap space or file system.

Table 3.3: Compressed sizes with respect to BAM (without taking into account the compressed sizes of the corresponding reference sequences), averaged over all items, and maximum RSS, additionally averaged over encoding and decoding. Note that the averages shown are not weighted. Hence, they present an average over the different data types (not the specific data items) that were used in the experiments.

Codec	∅ Size w.r.t. BAM	∅ Max. RSS / MiB
DeeZ	0.3655	6,774
gzip	0.9625	10
Samtools (BAM)	1	11
Scramble (CRAM 3.0)	0.3448	1,166
—w/o ref.	0.9976	783
TSC	0.8305	563

Our model consists of two main elements. The first main element is the use of a sliding window to track recently encoded reads. The second main element is to use the data tracked within the sliding window to infer a so-called local sliding consensus reference for the prediction of subsequent reads. These two elements have been adopted in the ISO/IEC 23092 series. In Section 3.3 we have shown how exactly these elements are integrated in ISO/IEC 23092-2:2020.

We coupled our model with generic compression methods to evaluate its performance with respect to the state of the art. For our evaluation, we made use of a previously published benchmark suite, that we co-authored. Our results show that TSC provides the best trade-off of memory usage and achieved compressed size. Among the specialized codecs, TSC exhibits the lowest average memory consumption of 563 MiB. At the same time, TSC reaches an average compressed size of 0.8305 with respect to BAM, where the other specialized non-reference-based codec (non-reference-based mode of Scramble) only reaches 0.9976.

4

COMPRESSION OF QUALITY SCORES

As outlined in Chapter 2, Section 2.4, the two-stage process of modeling and coding eliminates redundancy in the input data. However, the input data may also contain information that is considered irrelevant. This irrelevant information can be reduced by applying quantization. Regarding quantization, in all cases a trade-off must be made between the amount of loss that can be accepted after reconstruction of the data, and the compression that can be achieved.

As also outlined in Chapter 2 (Section 2.4.3.1), models may not only serve the purpose of redundancy reduction; they can also serve as proxy models that control other parts of the compression process, such as quantization.

Building on these two concepts (proxy models and quantization), in this chapter, we present two mathematical models that are used to control the quantization of quality scores: the genotype likelihood model and the activity-based posterior model. The activity-based posterior model is an extension of the genotype likelihood model. It therefore includes the genotype likelihood model. In short, both models predict the "importance" of each quality score. This importance is subsequently used to control the "coarseness" of the quantization that is applied to the quality scores.

This chapter is structured as follows. First, we review the state of the art in Section 4.1. Here, we also introduce our contribution CALQ [VOH17; VOH18]. In Section 4.2, we present the architecture of CALQ in full detail, with special emphasis on the two conceived models. The CALQ technology has been adopted in the ISO/IEC 23092 series. Hence, in Section 4.3, we detail how exactly CALQ is integrated in ISO/IEC 23092-2:2020. Following precedence in the literature, we designed an extensive experiment setup to quantify the impact of quality score quantization on downstream analyses, which we present in Section 4.4. We show and discuss the results in Section 4.5. Finally, we conclude our findings in Section 4.6.

4.1 STATE OF THE ART AND CONTRIBUTION

It has been shown that quality scores can take up to 80% of the lossless compressed size [Och+17]. This is mainly due to the large quality score alphabet that is usually used. For example, most Illumina sequencing machines emit 42 different quality scores. Addressing the issue of quality score compression, Illumina proposed a binning method to reduce the number of difference quality scores from 42 to 8. With this proposal, Illumina paved the way for lossy compression of quality scores.

The drawback of allowing lossy compression of quality scores is that downstream analyses could be affected. However, Yu et al. [Yu+15], Ochoa et al. [Och+17], as well as our own works [Alb+16; Her+17; Her+18] showed that quality scores compressed using more sophisticated methods may not only perform better than Illumina-binned quality scores in downstream analyses, but in some cases may even perform better than the original quality scores because these methods remove noise from the data. These conclusions were made by conducting rate-distortion analyses, where distortion metrics are applied that are specifically tailored to downstream analyses.

The most advanced lossy quality score compression methods that can be found in the literature are Crumble [BMD18], Quartz [Yu+15], and QVZ2 [HOW16].

Crumble [BMD18] uses a simple heterozygous consensus algorithm (taken from Gap5 [BW10]) to produce a consensus call for each locus, coupled with a confidence. If the calls are highly confident then the algorithm sets the corresponding quality scores to a fixed high or low value, depending on whether they agree with the call. In the case that the calls are not highly confident, the algorithm keeps the original quality scores. Crumble is, besides CALQ, the only method that is also operating on a per-locus basis, rather than on a per-read basis.

Quartz [Yu+15] quantizes quality scores by smoothing a large fraction of them based on their k-mer neighborhood in the nucleotide sequences. A prerequisite for the operation of Quartz is the creation of a dictionary of common k-mers for each species. Then, for a given set of reads, Quartz breaks these reads up into a set of overlapping so-called supporting k-mers. After that, each position in a supporting k-mer that differs from a dictionary k-mer is noted as a possible variant. Quartz assumes that these divergent nucleotides correspond to sequencing errors or to SNPs, respectively. The corresponding quality scores are preserved by Quartz, and other quality scores are set to a predefined default value.

QVZ2 [HOW16] models the quality scores in each read as a Markov process of order one. This approach is rooted in the finding that quality

scores are highly correlated with their neighbors. The transition probabilities of the Markov process are derived from the entire dataset to be compressed. Subsequently, these transition probabilities are used to compute a set of Lloyd-Max quantizers, indexed by the position within a read as well as by the previously quantized value. Finally, an adaptive arithmetic encoder is used to compress the quantized quality scores.

Our contribution to the field of compression of quality scores, CALQ, is centered around two mathematical models that are used to control the quantization of quality scores. The first model, the genotype likelihood model, infers the likelihood distribution of the genotype for each locus from the observable data, i.e., the read and alignment information, using a statistical model. The likelihood distribution of the genotype is further used to compute a "genotype uncertainty profile" over the loci. The second model, the activity-based posterior model, builds on the genotype likelihood model by adding a more realistic prior to infer the posterior distribution of the genotype for each locus. Incorporating further systematic assumptions, this posterior distribution of the genotype is used to eventually compute an "activity profile" over the loci. The output of either model (genotype uncertainty profile or activity profile, respectively) is used to determine the locus-wise acceptable level of distortion for the quality scores such that subsequent downstream analyses are presumably not affected. Finally, the quality scores are quantized accordingly. This way, high compression is achieved while at the same time only a negligible impact on downstream analyses is permitted.

4.2 CALQ ARCHITECTURE

In this section, we describe the CALQ architecture from the encoder point of view. The encoder input consists of alignments, i.e., reads, including quality scores, as well as alignment information. The encoder output is a compressed bitstream representing the quantized quality scores.

In a first step, CALQ uses either the genotype likelihood model or the activity-based posterior model to determine the genotype uncertainty profile or the activity profile, respectively, of which either is used in turn to infer the locus-wise acceptable level of distortion for the quality scores. More specifically, a quantizer index is computed for each locus. Each quantizer index identifies a specific precomputed quantizer to be used for the quantization of all quality scores at a specific locus.

In a second step, the quantizer indices are entropy-encoded and form the first part of the compressed bitstream that represents the quantized quality scores.

In a third step, the quality scores are quantized using the selected quantizers. This process yields one quantization index per quality score. Note that we make a distinction between *quantizer indices* (for the locus-wise quantizer selection) and *quantization indices* (output of quality score quantization). The quantization indices are also entropy-encoded and form the second and final part of the compressed bitstream.

Using the alignment information, a decoder is able to reconstruct the quantized quality scores using the quantizer indices (one for each locus) and the quantization indices (one for each quantized quality score).

CALQ includes two models: the genotype likelihood model and the—significantly more complex—activity-based posterior model. The basis in both models is the inference of the posterior distribution of the genotype.

The genotype likelihood model infers the likelihood distribution of the genotype for each locus from the observable data, i.e., the read and alignment information. Here, the most simple model is assumed for the prior distribution of the genotype (i.e., a "flat" prior). This leads to a direct proportionality between the likelihood and the posterior distributions of the genotype. The likelihood distribution of the genotype is further used to compute a "genotype uncertainty profile" over the loci. The genotype uncertainty profile is finally used to compute a quantizer index for each locus.

The activity-based posterior model builds on the genotype likelihood model by incorporating a more realistic model for the prior to infer the posterior distribution of the genotype for each locus. Hence, here, the posterior distribution of the genotype is used explicitly and referred to as "raw" activity profile. This model also incorporates further systematic assumptions to eventually compute a "smooth" activity profile over the loci. More specifically, information about high-quality softclips is used to refine the raw activity profile. Also, a low-pass filter is applied. Finally, the smoothed activity profile is used to compute a quantizer index for each locus. The activity-based posterior model draws inspiration from the GATK HaplotypeCaller [McK+10; DeP+11; VO20], where an activity profile is used to identify active regions that are subsequently screened in more detail for variants.

In summary, comparing the two models, the activity-based posterior model leads to a more conservative quantization of quality scores, which in turn leads to a slightly higher rate for the compressed quality scores. However, the hope here is that this more realistic model is able to minimize the impact of quantization on subsequent downstream analyses, thus leading to a more favorable trade-off between rate and distortion.

4.2.1 *Genotype Likelihood Model*

The first model, the genotype likelihood model, infers the likelihood distribution of the genotype for each locus from the observable data, i.e., the read and alignment information, using a statistical model. More specifically, the observable data at a specific locus are the nucleotides and the associated quality scores of all alignments that overlap the locus. The likelihood distribution of the genotype is further used to compute a "genotype uncertainty profile" over the loci. The genotype uncertainty profile is finally used to compute a quantizer index for each locus.

More concretely, the genotype uncertainty at a specific locus can be viewed as a metric that measures the probability that a unique genotype is the correct one. Hence, the genotype uncertainty can be used to determine the amount of loss that is acceptable in the quality scores at the corresponding locus: if a unique genotype is very likely, then a high loss in the quality scores might be acceptable, and vice versa.

We control the amount of introduced loss by using the genotype uncertainty profile to compute a specific quantizer index r for each locus. Each quantizer index r identifies a specific precomputed quantizer to be used for the quantization of all quality scores at that specific locus. Here, we use a set of seven uniform mid-tread quantizers with two to eight quantization levels, i.e., $r \in \{2, 3, \ldots, 8\}$.

Recall that a quantizer can be decomposed into two distinct stages, which are referred to as the quantization stage and the reconstruction stage (see Section 2.4.5). The quantization stage maps the input value (here: quality score q) to an integer quantization index k. The reconstruction stage maps the quantization index k to the reconstructed quality score $\breve{q}_k$.

In other words, the amount of introduced loss is parametrized by a quantizer index r, which represents a quantizer with r quantization levels. Specifically, if the genotype likelihood models infers that two or more different genotypes are likely to be true, then the genotype uncertainty will be high and hence, r will be high. However, if there is enough evidence in the data that a particular genotype is likely the correct one, then the genotype uncertainty will be low, and therefore r will be low. Consequently, the achievable compression of the quality scores associated to a particular locus is driven by the genotype uncertainty at that locus.

This idea is depicted in Figure 4.1. The figure shows at the top four aligned nucleotides sequences, including the associated quality scores. Recall that a quality score is a value that indicates the confidence in a base call (see Section 2.3.2). Quality scores are logarithmically linked to the base-calling error probability. Also, they are typically offset to be able

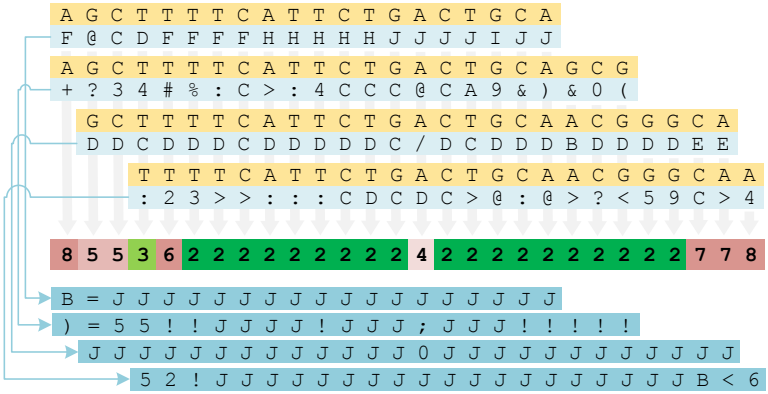


Figure 4.1: Genotype uncertainty inference. The figure shows at the top four aligned nucleotide sequences (yellow), including the associated quality scores (light blue). The red and green colored bar in the middle represents the inferred genotype uncertainty at each locus. Dark green represents low uncertainty, while dark red represents high uncertainty. The numbers shown in the middle bar are the quantizer indices r chosen for each locus. The value of r also indicates the number of different quantization indices of its associated quantizer. Finally, the figure shows at the bottom the reconstructed quality scores. This figure was already presented in [VOH18].

to represent them with a single ASCII character. Hence, in Figure 4.1, quality scores represented by special characters or digits correspond to high uncertainties on the genotype (i.e., low qualities); quality scores represented by uppercase letters correspond to low genotype uncertainties (i.e., high qualities). For simplicity, let us assume that the reads originate from a haploid genome. The bar in the middle of the figure, immediately below the reads, shows with colors the inferred genotype uncertainty at each locus. Dark green represents low uncertainty, while dark red represents high uncertainty. The genotype uncertainty is then used to infer a quantizer index r, representing a quantizer with r quantization levels. The corresponding quantizer indices are shown as numbers in the middle bar. In other words, each number in the middle bar indicates the number of quantization indices that will be used to represent the quality scores at the corresponding locus. Recall that a large number is associated with a high genotype uncertainty, and vice versa. Finally, the figure shows at the bottom the reconstructed quality scores.

To illustrate the concept with an example, let us focus on the left-most locus in Figure 4.1. At this locus there is not enough evidence in the

data to support a particular genotype, because only two nucleotides cover that locus, with one of them having a rather high quality score ("F"), and the other one of them having a rather low quality score ("+"). Therefore, a large quantizer index r will be inferred, namely $r = 8$ in this example. Conversely, Figure 4.1 contains a number of loci with low genotype uncertainty (dark green parts of the bar in the middle). Here, only $r = 2$ quantization indices will be used to represent all quality scores at these loci. Finally, note that at the fifth locus all nucleotides align perfectly. (Recall that we assume a haploid genome.) However, since half of the corresponding quality scores at that particular locus are low, the genotype uncertainty is high, resulting in $r = 6$ quantization indices to be used.

The result of the encoding algorithm is a series of quantizer indices (one quantizer index per locus), and, for each alignment, a vector of quantization indices. Both types of information undergo entropy encoding.

In the following sections, we elaborate in more detail on the inference of the genotype uncertainty (Section 4.2.1.1) and the quantizer design (Section 4.2.1.2).

4.2.1.1 *Inference of the Genotype Uncertainty*

For any particular locus, we represent the genotype by a discrete random variable $\mathbf{g}$ drawn from the genotype alphabet $\mathbb{G}$. We express the genotype as vector of alleles, which we also represent using discrete random variables, i.e., $\mathbf{g} = (a_0 a_1 \cdots a_{H-1})^T$, where each allele is drawn from the allele alphabet $\mathbb{A}$. The number of alleles that make up a genotype, H, is the ploidy of the species.

We can derive the number of possible genotypes, i.e., the cardinality of the genotype alphabet, by computing all possible allele combinations with repetitions:

$$|\mathbb{G}| = \binom{|\mathbb{A}| + H - 1}{|\mathbb{A}| - 1}. \tag{4.1}$$

As an example, for the allele alphabet $\mathbb{A} = \{A, C, G, T\}$ with cardinality $|\mathbb{A}| = 4$ and the case of a diploid organism with $H = 2$ this would result in $|\mathbb{G}| = 10$ possible genotypes.

Recalling the example of Figure 4.1, let us consider a set of sorted and aligned nucleotide sequences, including the associated quality scores. We denote by D the number of nucleotides covering a particular locus, i.e., the depth of the pileup. (D is also the number of quality scores covering the locus.) Let n_d be the nucleotide from the d-th nucleotide sequence covering the locus, and let q_d be the value of the corresponding Phred quality score, i.e., without offset (see Equation 2.1).

The goal here is to compute the relative posterior distribution of the genotype $\mathbf{g}$ given the evidence, i.e., the observable nucleotides

$$\mathbf{n} = (n_0 n_1 \cdots n_d \cdots n_{D-1})^T, \tag{4.2}$$

parametrized by the observable quality scores

$$\mathbf{q} = (q_0 q_1 \cdots q_d \cdots q_{D-1})^T. \tag{4.3}$$

The absolute posterior distribution of the genotype $\mathbf{g}$ is defined as

$$P(\mathbf{g} = \mathbf{g} \,|\, \mathbf{n}; \mathbf{q}) = \frac{P(\mathbf{n} \,|\, \mathbf{g} = \mathbf{g}; \mathbf{q}) \cdot P(\mathbf{g} = \mathbf{g})}{P(\mathbf{n})}. \tag{4.4}$$

The model evidence $P(\mathbf{n})$ is the same for all possible genotypes. Therefore, this factor need not be considered when determining the *relative* posterior distribution of the genotype. The relative posterior distribution is therefore proportional to the likelihood times the prior:

$$P(\mathbf{g} = \mathbf{g} \,|\, \mathbf{n}; \mathbf{q}) \propto P(\mathbf{n} \,|\, \mathbf{g} = \mathbf{g}; \mathbf{q}) \cdot P(\mathbf{g} = \mathbf{g}). \tag{4.5}$$

In the genotype likelihood model, we use the most simple prior, i.e., a "flat" prior. We assume that all genotypes are equally probable:

$$P(\mathbf{g} = \mathbf{g}) = \frac{1}{|G|} \;\forall\; \mathbf{g}. \tag{4.6}$$

Therefore, the posterior is directly proportional to the likelihood:

$$P(\mathbf{g} = \mathbf{g} \,|\, \mathbf{n}; \mathbf{q}) \propto P(\mathbf{n} \,|\, \mathbf{g} = \mathbf{g}; \mathbf{q}). \tag{4.7}$$

Consequently, in the genotype likelihood model, it is sufficient to compute the likelihood distribution of the genotype, which is given by

$$P(\mathbf{n} \,|\, \mathbf{g} = \mathbf{g}; \mathbf{q}) = \prod_{d=0}^{D-1} P(n_d \,|\, \mathbf{g} = \mathbf{g}; q_d), \tag{4.8}$$

where $P(n_d \,|\, \mathbf{g} = \mathbf{g}; q_d)$ is the likelihood of having observed n_d given that the genotype was $\mathbf{g}$. Recall that the genotype is expressed as vector of alleles. Now note that each nucleotide n_d was drawn from the alleles. Without further information we hence assign to all alleles an equal probability of being the one from which the nucleotide n_d was drawn. Therefore, the likelihood of having observed n_d given that the genotype was $\mathbf{g}$ is given by

$$P(n_d \,|\, \mathbf{g} = \mathbf{g}; q_d) = \sum_{h=0}^{H-1} \frac{P(n_d \,|\, a_h = a_h; q_d)}{H}, \tag{4.9}$$

where $P(n_d \mid a_h = a_h; q_d)$ is the likelihood of having observed n_d given the assumption that the true nucleotide was the allele a_h, parametrized by q_d. This probability is given by

$$P(n_d \mid a_h = a_h; q_d) = \begin{cases} 1 - 10^{-\frac{q_d}{10}}, & n_d = a_h, \\ \frac{10^{-\frac{q_d}{10}}}{|\mathbb{A}|-1}, & n_d \neq a_h. \end{cases} \tag{4.10}$$

Finally, given the likelihood $P(\mathbf{n} \mid \mathbf{g} = \mathbf{g}; \mathbf{q})$, the genotype uncertainty u is calculated by applying a metric m:

$$u = m\left(P(\mathbf{n} \mid \mathbf{g} = \mathbf{g}; \mathbf{q})\right). \tag{4.11}$$

We elect m to be one minus the difference between the maximum likelihood and the second largest likelihood:

$$m = 1 - (P_{max} - P_2), \tag{4.12}$$

with

$$P_{max} = \max_{\mathbf{g}} P(\mathbf{n} \mid \mathbf{g} = \mathbf{g}; \mathbf{q}) \tag{4.13}$$

and

$$P_2 = \max_{\mathbf{g}, \mathbf{g} \neq \mathbf{g}_{max}} P(\mathbf{n} \mid \mathbf{g} = \mathbf{g}; \mathbf{q}) \tag{4.14}$$

with

$$\mathbf{g}_{max} = \operatorname*{argmax}_{\mathbf{g}} P(\mathbf{n} \mid \mathbf{g} = \mathbf{g}; \mathbf{q}). \tag{4.15}$$

Note that any other metric, such as the entropy (see Equation 2.17), could also be used. However, we chose the metric shown in Equation 4.12, because it provides more meaningful results when the likelihood consists of only a few approximately equally likely genotypes.

4.2.1.2 *Quantizer Design*

The genotype uncertainty u is used to compute the quantizer index r as

$$r = f(u), \tag{4.16}$$

where f is a monotonous increasing function. It maps the possible genotype uncertainty values to an integer set of quantizer indices. Each quantizer index identifies a specific precomputed quantizer to be used for the quantization of all quality scores at the current locus. Here, we use a set of seven uniform mid-tread quantizers with two to eight quantization levels, i.e., $r \in \{2, 3, \ldots, 8\}$. Hence, we configure f to also be a uniform mid-tread quantizer that outputs integer values in the set $\{2, 3, \ldots, 8\}$. Figure 4.2 depicts the characteristic of f.

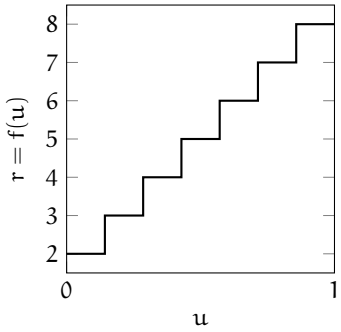


Figure 4.2: Mapping of genotype uncertainty to quantizer index.

We opted for two quantization indices for the "coarsest" quantizer, as this binary decision is well qualified for loci with low genotype uncertainty. Given the trend to downscale the quality score resolution, we chose eight quantization indices for the "finest" quantizer to mimic Illumina's 8-binning. In other words, we use the 8-binning scheme as a baseline. At the same time we leave room for improvements of both the resulting compression as well as the downstream analysis performance by selecting quantizers with fewer quantization indices when appropriate.

4.2.2 *Activity-Based Posterior Model*

The second model, the activity-based posterior model, builds on the genotype likelihood model by adding a more realistic prior to infer the relative posterior distribution of the genotype for each locus. Incorporating further systematic assumptions, this relative posterior distribution of the genotype is used to eventually compute an "activity profile" over the loci.

As the first step, we define the raw activity profile—using the relative posterior distribution of the genotype for all loci—as follows:

$$a_{\text{raw}}(l) := \left(1 - \max_{\mathbf{g}} P(\mathbf{g} = \mathbf{g} \mid \mathbf{n}; \mathbf{q})\right)\Big|_{l} \quad \forall\, l. \tag{4.17}$$

In the case that at a specific locus l the maximum relative posterior probability is low, then there will be a high activity. In turn, if the maximum relative posterior probability is high, then the activity will be low.

4.2.2.1 *Reference-Based Model for the Genotype Prior*

Recall that the relative posterior distribution is proportional to the likelihood times the prior (see Equation 4.5). In the genotype likelihood model a flat prior was used (see Equation 4.6). Here, in the activity-based posterior model, to define a more realistic prior, we make use of an external reference sequence. For this purpose, we group the possible genotypes into $H+1$ groups. Recall that we express the genotype as a vector-valued random variable $\mathbf{g} = (a_0 a_1 \cdots a_{H-1})^T$ consisting of H alleles.

The first group γ_0 contains the single genotype in which all alleles match the reference. We refer to group γ_0 as the homozygous-reference group. The next groups γ_η, with $\eta \in \mathbb{N} \setminus \{0\}$ and $\eta < H$, contain all those genotypes in which η alleles do not match the reference. We refer to the groups γ_η as the heterozygous-variant groups. The last group γ_H contains all those genotypes in which none of the alleles match the reference. We refer to group γ_H as the homozygous-variant group.

As an example, consider the allele alphabet $\mathbb{A} = \{A, C, G, T\}$ with cardinality $|\mathbb{A}| = 4$ and the case of a diploid organism with $H = 2$. Consequently, the genotype alphabet $\mathbb{G}$ consists of the following $|\mathbb{G}| = \binom{|\mathbb{A}|+H-1}{|\mathbb{A}|-1} = 10$ genotypes[1]:

$$\begin{aligned}\mathbb{G} &= \{\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4, \mathbf{g}_5, \mathbf{g}_6, \mathbf{g}_7, \mathbf{g}_8, \mathbf{g}_9\} \\ &= \{(A,A), (A,C), (A,G), (A,T), (C,C), \\ &\qquad (C,G), (C,T), (G,G), (G,T), (T,T)\}.\end{aligned} \tag{4.18}$$

Let us assume that the reference contains the symbol A at the locus under consideration. The homozygous-reference group γ_0 hence consists of genotype $\mathbf{g}_0 = (A, A)$:

$$\gamma_0 = \{\mathbf{g}_0\} = \{(A,A)\}. \tag{4.19}$$

In this example, because we considered a ploidy of $H = 2$, there is only one heterozygous-variant group, the group γ_1, containing those genotypes in which exactly one allele does not match the reference:

$$\gamma_1 = \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\}. \tag{4.20}$$

The third and last group γ_2—the homozygous-variant group—contains all those genotypes where none of the alleles match the reference. In this example, these are all those genotypes that are not already assigned to γ_0 or γ_1:

$$\gamma_2 = \mathbb{G} \setminus (\gamma_0 \cup \gamma_1) = \{\mathbf{g}_4, \ldots, \mathbf{g}_9\}. \tag{4.21}$$

1 See Equation 4.1 for the definition of the genotype alphabet cardinality.

In summary, our prior model consists of the $(H+1)\cdot|\mathbb{A}|$ group probabilities. (The $(H+1)$ group probabilities need to be computed for each possible reference allele.) Each group probability is computed as the sum of the individual genotype probabilities that correspond to each group. In the example above, the model would hence consist of $(H+1)\cdot|\mathbb{A}| = (2+1)\cdot 4 = 12$ probabilities.

To actually determine the group probabilities, we assume that a variant occurs every 1,000 loci, according to [DeP+11]. Hence, the probability for exactly one deviation from the reference is:

$$P_1 = \frac{1}{1000}. \tag{4.22}$$

Furthermore, also according to [DeP+11], we assume that the probability for $v > 1$ deviations from the reference is:

$$P_v = \frac{P_1}{v}. \tag{4.23}$$

Finally, the probability that all alleles match the reference can be computed as complementary probability to P_1 and all P_v:

$$P_0 = 1 - \left(P_1 + \sum_{v=2}^{H} \frac{P_1}{v}\right) = 1 - \sum_{v=1}^{H} \frac{P_1}{v}. \tag{4.24}$$

4.2.2.2 *Impact of High-Quality Softclips*

In general, a high abundance of high-quality softclips is an indication for misalignments. Hence, the idea here is to increase the raw activity profile (see Equation 4.17) at loci that are surrounded by high-quality softclips.

Let s_d be the number of high-quality softclips in read d. Similar to [DeP+11], we define high-quality softclips as those softclips that are associated to Phred quality scores $q > q_{HQ}$ with $q_{HQ} = 29$.

The average number of high-quality softclips in all reads covering locus l is:

$$\bar{s}(l) = \frac{1}{D} \sum_{d=0}^{D-1} s_d. \tag{4.25}$$

If a locus l is associated to a high average number $\bar{s}_{thresh}$ of high-quality softclips, then it is considered to be particularly active, and then l is added to the set S of active high-quality softclip loci. In analogy to [DeP+11], we use $\bar{s}_{thresh} = 7$.

Next, we need to infer for a specific locus λ the subset $S|_\lambda$ of high-quality softclips that impact the activity profile at this locus. We do this

by relating the average number of high-quality softclips to the distance from the locus under consideration:

$$\mathcal{S}\big|_{\lambda} : \{l \in \mathcal{S} \mid \bar{s}(l) > |l - \lambda|\}. \tag{4.26}$$

Given the raw activity profile $a_{raw}(l)\big|_{l=\lambda}$ at locus λ and the set $\mathcal{S}\big|_{\lambda}$ of active high-quality softclip loci with an impact on locus λ, we can finally compute the softclip-adjusted activity profile value at locus λ as

$$a_{sc}(\lambda) = a_{raw}(l)\big|_{l=\lambda} + \sum_{l \in \mathcal{S}|_{\lambda}} a_{raw}(l). \tag{4.27}$$

Consequently, we obtain the entire softclip-adjusted activity profile as

$$a_{sc}(l) := a_{sc}(\lambda)\big|_{\lambda=l} \; \forall \; l. \tag{4.28}$$

4.2.2.3 *Normalization of the Softclip-Adjusted Activity Profile*

Recall that the raw activity profile is defined using the maximum of the relative posterior distribution of the genotype for all loci (see Equation 4.17). The particular genotype that is associated to the maximum at a specific locus is defined as

$$\mathbf{g}_{max} = \underset{\mathbf{g}}{\operatorname{argmax}}\, P(\mathbf{g} = \mathbf{g} \mid \mathbf{n}; \mathbf{q}). \tag{4.29}$$

Further, we can in general write the raw activity profile as being parametrized by the genotype: $a_{raw}(l; \mathbf{g})$.

We can also discern that the raw activity profile contains probabilities, i.e.,

$$a_{raw}(l; \mathbf{g})\big|_{\mathbf{g}=\mathbf{g}_{max}} + a_{raw}(l; \mathbf{g})\big|_{\mathbf{g}=\neg\mathbf{g}_{max}} = 1. \tag{4.30}$$

However, in contrast to the raw activity profile, the softclip-adjusted activity profile $a_{sc}(l)$—which we can also write as being parametrized by the genotype as $a_{sc}(l; \mathbf{g})$—does not contain probabilities, because of the addition of raw activities from near high-quality softclip loci. Hence,

$$a_{sc}(l; \mathbf{g})\big|_{\mathbf{g}=\mathbf{g}_{max}} + a_{sc}(l; \mathbf{g})\big|_{\mathbf{g}=\neg\mathbf{g}_{max}} \neq 1. \tag{4.31}$$

In the following text we will abbreviate $a_{raw}(l; \mathbf{g})\big|_{\mathbf{g}=\mathbf{g}_{max}}$ as $a_{raw}(l)\big|_{\mathbf{g}_{max}}$. We will abbreviate $a_{raw}(l; \mathbf{g})\big|_{\mathbf{g}=\neg\mathbf{g}_{max}}$, $a_{sc}(l; \mathbf{g})\big|_{\mathbf{g}=\mathbf{g}_{max}}$, $a_{sc}(l; \mathbf{g})\big|_{\mathbf{g}=\neg\mathbf{g}_{max}}$ analogously as $a_{raw}(l)\big|_{\neg\mathbf{g}_{max}}$, $a_{sc}(l)\big|_{\mathbf{g}_{max}}$, $a_{sc}(l)\big|_{\neg\mathbf{g}_{max}}$.

A usual method to normalize signals such as the softclip-adjusted activity profile is the use of the softmax function. However, the softmax function would excessively distort the softclip-adjusted activity profile.

Instead, we use the observation that, by adding raw activities from near high-quality softclip loci, only $a_{raw}(l)\big|_{g_{max}}$ can grow, but $a_{raw}(l)\big|_{\neg g_{max}}$ cannot. Hence, we can conclude that

$$a_{sc}(l)\big|_{\neg g_{max}} = a_{raw}(l)\big|_{\neg g_{max}}. \tag{4.32}$$

We can therefore normalize the softclip-adjusted activities using the sum $a_{sc}(l)\big|_{g_{max}} + a_{raw}(l)\big|_{\neg g_{max}}$:

$$\frac{a_{sc}(l)\big|_{g_{max}}}{a_{sc}(l)\big|_{g_{max}} + a_{raw}(l)\big|_{\neg g_{max}}} + \frac{a_{raw}(l)\big|_{\neg g_{max}}}{a_{sc}(l)\big|_{g_{max}} + a_{raw}(l)\big|_{\neg g_{max}}} = 1. \tag{4.33}$$

Finally, we can define the normalized activity profile:

$$a_{norm}(l) := \frac{a_{sc}(l)\big|_{g_{max}}}{a_{sc}(l)\big|_{g_{max}} + a_{raw}(l)\big|_{\neg g_{max}}}. \tag{4.34}$$

As an example, consider the flat raw activity profile

$$a_{raw}(l)\big|_{g_{max}} = a_{raw}(l)\big|_{\neg g_{max}} = \hat{a}_{raw} = \frac{1}{2} \;\forall\, l. \tag{4.35}$$

We can now write the softclip-adjusted activity profile, parametrized by the number n of active high-quality softclips with an impact on locus l, as

$$a_{sc}(l; n) = \hat{a}_{raw} + n \cdot \hat{a}_{raw}. \tag{4.36}$$

We can then write the normalized activity profile as

$$\begin{aligned} a_{norm}(l; n) &= \frac{a_{sc}(l; n)}{a_{sc}(l; n) + a_{raw}(l)\big|_{\neg g_{max}}} && (4.37)\\ &= \frac{\hat{a}_{raw} + n \cdot \hat{a}_{raw}}{\hat{a}_{raw} + n \cdot \hat{a}_{raw} + \hat{a}_{raw}} && (4.38)\\ &= \frac{1+n}{2+n} = a_{norm}(n) \;\forall\, l. && (4.39) \end{aligned}$$

Figure 4.3 visualizes this example normalized activity profile. The figure shows that activities that have not been impacted by high-quality softclips remain unaffected ($n = 0$). High-quality softclips increase the activity, but the activity is constrained such that it never becomes 1.

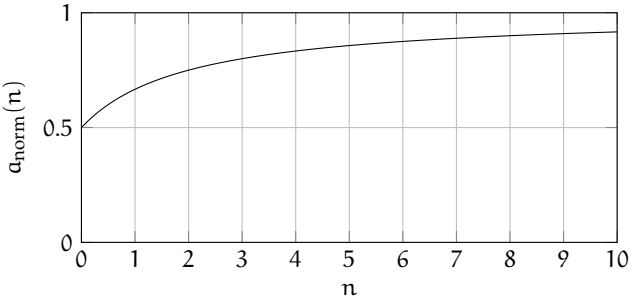


Figure 4.3: Visualization of parametrized normalized activity.

4.2.2.4 *Filtering the Normalized Activity Profile*

As a last step, the normalized activity profile $a_{norm}(l)$ is smoothed by applying a Gaussian filter. In other words, the normalized activity profile $a_{norm}(l)$ is convolved with the Gaussian function, parametrized by its standard deviation σ, to obtain the filtered activity profile

$$a_{filtered}(l;\sigma) := g(l;\sigma) * a_{norm}(l), \tag{4.40}$$

with

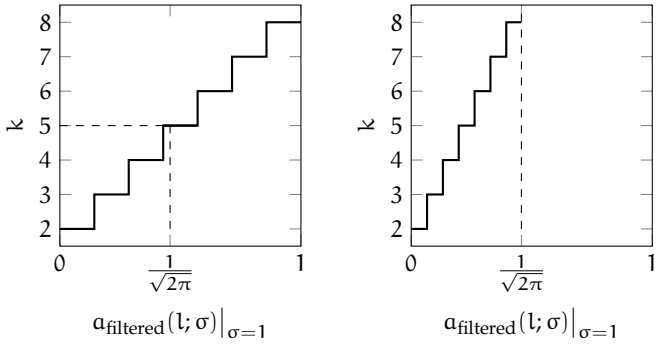
$$g(x;\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}. \tag{4.41}$$

To filter the normalized activity profile, which exhibits a discrete domain, we use a sampled Gaussian kernel that is produced by sampling points from the continuous Gaussian function. Moreover, convolution with the Gaussian function theoretically requires an infinite window length. Since the Gaussian function decays swiftly, it is reasonable to truncate the filter window. Here we use a filter window size of $W + 1$ and compute the filtered activity profile as:

$$a_{filtered}(l;\sigma) = \sum_{i=-W}^{W} a_{norm}(l-i) \cdot g(i;\sigma). \tag{4.42}$$

Specifically, we use $W = 23$. This value was derived by using the standard deviation $\sigma = 17$, the default value used in GATK, and the value $\delta = 0.01$, specifying the value of the Gaussian function that is deemed negligible:

$$\delta < \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{W^2}{2\sigma^2}} \Rightarrow W > \sqrt{-2\sigma^2 \cdot \ln\left(\delta\sqrt{2\pi}\sigma\right)}. \tag{4.43}$$



(a) Original quantizer characteristic. (b) Scaled quantizer characteristic.

Figure 4.4: Original and scaled quantizer characteristics.

4.2.2.5 *Quantizer Design*

The quantizer design for the activity-based posterior model is analogous to the quantizer design in the genotype likelihood model (see Section 4.2.1.2). However, here, two modifications are applied.

In the genotype likelihood model, the genotype uncertainty is used to compute the quantizer index (see Equation 4.16). In contrast, here, the filtered activity profile $a_{\text{filtered}}(l;\sigma)$ is used to compute the quantizer index r as

$$r = f\left(a_{\text{filtered}}(l;\sigma)\big|_{\sigma=\sigma^*}\right), \tag{4.44}$$

for fixed values σ^* of σ.

Also, in the genotype likelihood model, f is configured to be a uniform mid-tread quantizer that outputs integer values in the set $\{2, 3, \ldots, 8\}$ (see Figure 4.2). However, here, the filtering of the normalized activity profile "spreads" the filtered activity profile. As a consequence, high quantizer indices are not reached anymore. Consider for example the filtering with $\sigma = 1$. In this case, the highest possible value of $a_{\text{filtered}}(l;\sigma)\big|_{\sigma=1}$ is $\frac{1}{\sqrt{2\pi}}$. As revealed in Figure 4.4a, the quantizer indices $k = 6$ to $k = 8$ would not be used. As a solution, we re-scale the quantizer characteristic using $\frac{1}{\sqrt{2\pi}\sigma}$. Figure 4.4b shows the resulting scaled quantizer characteristic pertaining to the example.

4.2.3 *Entropy Coding*

The result of the encoding algorithm (using either model) is a vector of quantizer indices (one quantizer index per locus) and, for each align-

ment, a vector of quantization indices, where each quantization index corresponds to a particular quality score.

First, the quantizer index vector is entropy-encoded using a range coder. Recall that range coding is a special case of arithmetic coding (see Section 2.4.4.3).

Second, and finally, the quantization indices are regrouped into seven groups, where each group corresponds to a specific quantizer from the set of precomputed quantizers [VOa; VOb]. Each group of quantization indices is then entropy-encoded using a separate range coder for each group[2]. The regrouping facilitates a more efficient entropy encoding of the quantization indices, since the alphabets exhibit more non-uniform distributions. For example, the first group of quantization indices, corresponding to the quantizer with $r = 2$ quantization levels, will only contain two quantization indices.

Note that the decoder is agnostic to how the quantizer index vector is generated. This renders the CALQ architecture generic, i.e., arbitrary models (other than the genotype likelihood model or the activity-based posterior model) may be used for the generation of the quantizer index vector. We illustrate the generic decoder design and its integration into ISO/IEC 23092-2:2020 in Section 4.3.

4.3 INTEGRATION OF CALQ IN MPEG-G

The CALQ technology was submitted to the standardization process of the ISO/IEC 23092 series, and it is now integrated in its part 2. To illustrate this, we show and explain parts of the relevant decoding processes specified in ISO/IEC 23092-2:2020.

Figure 4.5 shows the top-level quality score decoding process, as specified in Clause 10.4.16.2 of ISO/IEC 23092-2:2020. This top-level process specifies the decoding of all quality scores belonging to a specific ISO/IEC 23092-2 record.

First, the top-level quality score decoding process decodes codebook indices (i.e., quantizer indices) by calling another decoding process (line 3). Figure 4.6 shows this quality score codebook index decoding process, as specified in Clause 10.4.1.3 of ISO/IEC 23092-2:2020. It is quite straightforward and needs no further explanation.

Second, and finally, the top-level quality score decoding process (Figure 4.5) decodes the actual quality scores (lines 6–18). An ISO/IEC 23092-2 record may, in general, contain multiple segments. For example, in the case of paired reads, each ISO/IEC 23092-2 record would contain two

2 It is also possible to use a single range coder, of which the internal model is reset for the encoding of each group.

```
1  decode_quality_values() {
2      if (qvCodebookIndexesLoadFlag == 1) {
3          decode_qv_codebook_indexes()
4          qvCodebookIndexesLoadFlag = 0
5      }
6      for (tSeg = 0; tSeg < numberOfRecordSegments; tSeg++) {
7          for (qs = 0; qs < qv_depth; qs++) {
8              ...
9              if (qvPresentFlag == 1) {
10                 decode_qvs()
11                 qvString = ""
12                 ...
13                 qualityValues[tSeg][qs] = qvString
14             } else {
15                 qualityValues[tSeg][qs] = ""
16             }
17         }
18     }
19 }
```

Figure 4.5: Top-level quality score decoding process as specified in Clause 10.4.16.2 of ISO/IEC 23092-2:2020. Note that the term "quality value" is used instead of "quality score". The dots in lines 8 and 12 indicate additional process parts, which are present in ISO/IEC 23092-2:2020, but which are not repeated here for brevity.

segments. Also, in an ISO/IEC 23092-2 record, each nucleotide may be associated to multiple quality scores[3]. Hence, the top-level quality score decoding process loops over the segments (line 6), as well as over the different associated quality scores (line 7). According to ISO/IEC 23092-2:2020, including quality scores in ISO/IEC 23092-2 records is optional. Hence, next, the decoding process checks whether quality scores are present in the subject ISO/IEC 23092-2 record (line 9). Then, in the case that quality scores are present, the actual quality score decoding process is called (line 10).

Figure 4.7 shows the quality score decoding process, as specified in Clause 10.4.1.3 of ISO/IEC 23092-2:2020.

The quality score decoding process loops over all nucleotides (bases) in a segment (line 2). First, for each nucleotide, the codebook to be used is identified. This, i.e., the retrieval of the respective codebook identifier (i.e., quantizer index) is shown in lines 3–11. There are a few special cases[4]

3 In the case that, for example, a recalibration of quality scores is applied to a set of alignments, the non-recalibrated quality scores may be preserved using this mechanism.

4 For example, classes I and HM may contain unaligned nucleotides, for which ISO/IEC 23092-2:2020 specifies the use of the "last" codebook (see line 4 in Figure 4.7).

```
1 decode_qv_codebook_indexes() {
2     if (qvNumCodebooksAligned > 1) {
3         pos = 0
4         for (j14,1 = 0; j14,1 < Size(subsequence1[]); j14,1++) {
5             qvCodebookIds[pos] = subsequence1[j14,1]
6             pos++
7         }
8     }
9 }
```

Figure 4.6: Quality score codebook index decoding process as specified in Clause 10.4.1.3 of ISO/IEC 23092-2:2020.

that need to be taken care of in order to retrieve the correct codebook identifier for a particular nucleotide. However, in the case of aligned data, there is a general case, which is shown in line 8. Here, an auxiliary array named "`basePos`" is used. Its objective is to map intra-segment base indices (represented by the variable named "`baseIdx`") to mapping positions (relative to the access unit start position). The actual quality score is then decoded (i.e., reconstructed[5]) in line 16, using the selected codebook.

4.4 EXPERIMENT SETUP

Since we are discussing the quantization of quality scores, the reconstructed quality scores can be different from the original ones. Hence, it is of primary importance to assess the effect that this loss has on downstream applications. We elect variant calling as a representative downstream application here because it is critical to clinical decision making and therefore widely used.

Specifically, we selected three different variant calling pipelines for our evaluation. The first pipeline is composed according to the GATK best practices workflow for germline short variant discovery [DeP+11]. The last step of this pipeline consists of filtering the called variants to remove false positives. The GATK best practices workflow specifies the use of VQSR; however, here we also consider the more basic "hard filtering" as the second pipeline. The third pipeline involves the variant caller

5 This is an implementation of the reconstruction stage, introduced in Section 2.4.5, where an index k (here: the variable named "`qvIndex`") is mapped to the corresponding reconstruction value $\check{x}_k$ (here: the variable named "`qualityValues`") using a LUT (here: the encoding parameter named "`qv_recon`") which maps each index to a corresponding reconstruction value.

```
decode_qvs() {
    for (baseIdx = 0; baseIdx < numBases; baseIdx++) {
        if ((classId == CLASS_I || classId == CLASS_HM) && !isAligned(
            ↪ baseIdx)) {
            qvCodeBookId = qv_num_codebooks_total - 1
        } else if (classId == CLASS_U) {
            qvCodeBookId = 0
        } else if (qvNumCodebooksAligned > 1) {
            qvCodeBookId = qvCodeBookIds[basePos[baseIdx]]
        } else {
            qvCodeBookId = 0
        }
        qvCodeBookSubSeq = qvCodeBookId + 2
        j = j14,qvCodeBookSubSeq
        j14,qvCodeBookSubSeq++
        qvIndex = decoded_symbols[14][qvCodeBookSubSeq][j]
        qualityValues[tSeq][qs][baseIdx] = qv_recon[qvCodeBookId][
            ↪ qvIndex]
    }
}
```

Figure 4.7: Quality score decoding process as specified in Clause 10.4.1.3 of ISO/IEC 23092-2:2020.

Platypus [Rim+14]. For further details on these pipelines we refer the reader to [VOH18].

After each run of each pipeline, a set of variant calls ("call set") is obtained, which is ultimately compared to a consensus variant call set ("consensus set"). Note that when using the first pipeline (GATK with VQSR), each variant is annotated with a VQSLOD score that provides an estimate of the probability that the variant is true. When using this pipeline, we use the VQSLOD score to further filter the call set. The filtering is done by first specifying a filter level, given as a percentage, e.g., 99.9%. Second, the filtering process then determines the value of the VQSLOD score above which 99.9% of the variants in the consensus set are included in the call set. Third, and finally, the determined VQSLOD score value is used as a threshold to filter the call set, where all variants below the threshold are discarded. Specifically, and similar to [Och+17] as well as [Alb+16], we have chosen to use four different levels of filtering, namely 90%, 99%, 99.9%, and 100%.

The outputs of all pipelines are analyzed with the Haplotype Comparison Tools[6] to benchmark the call sets obtained by each pipeline against a consensus set [Kru+19]. Note that, to make the benchmark more mean-

6 https://github.com/Illumina/hap.py

ingful, the comparison is restricted to the high-confidence regions of the consensus set. Again, for more details on these pipelines we refer the reader to [VOH18]. Note that in a few cases[7] (1.7%) the variant calling was not successful, due to unidentifiable reasons.

The comparison of the call set to the consensus set yields the following values:

- True Positives (TP): All those variants that are both in the consensus set and in the call set.
- False Positives (FP): All those variants that are in the call set but not in the consensus set.
- False Negatives (FN): All those variants that are in the consensus set but not in the call set.
- Non-assessed calls: All those variants that fall outside of the high-confidence regions of the consensus set. Note that previous works ([Mal+15; OHW15; Alb+16; Yu+15]) did not consider this distinction. In these previous works, all those variants were accounted as positives, and they were therefore inflating the false positives metric. This could be the cause for the weak performance of the variant callers in the aforementioned works.

These values are used to compute the precision P, i.e., the proportion of called consensus variants with respect to all called variants,

$$P = \frac{TP}{(TP + FP)}, \tag{4.45}$$

as well the recall R, i.e., the proportion of called consensus variants with respect to all consensus variants,

$$R = \frac{TP}{(TP + FN)}. \tag{4.46}$$

To measure the overall accuracy of each call set, we use the F_1 score, which is defined as the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}. \tag{4.47}$$

The sequencing data used for this analysis pertains to the same individual, namely NA12878. The reason behind this choice is that the NIST has released a consensus set of variants for this individual [Zoo+14]. Table 4.1 shows the details of the selected data. Following the approach

of [Och+17] we consider chromosomes 11 and 20. In addition, to broaden our test set, we also consider chromosome 3.

We compare our genotype likelihood model ("CALQ G") and our activity-based posterior model ("CALQ A-σ") to the state-of-the-art lossy quality score compressors Crumble, Quartz, and QVZ2. Also, we compare the models to Illumina's 8-level binning ("IL8B").

We evaluate the activity-based posterior model for three different filtering configurations, i.e., we filter the normalized activity profile using $\sigma = 10$ ("CALQ A-10"), $\sigma = 17$ ("CALQ A-17"), and $\sigma = 25$ ("CALQ A-25").

In the case of Crumble, we first performed the quantization using the Crumble software[8], version 0.5. Second, the resulting BAM file was compressed using Scramble, version 1.14.6. The compressed size of the quality scores in the resulting CRAM files was determined using the tool cram_size which is included in Scramble. We used Crumble in two different modes, "Crumble -1" and "Crumble -9", where the latter mode uses a more aggressive binning strategy.

We used Quartz, version 0.2.2[9]. For the working of Quartz, a sequence dictionary[10] is necessary. First, we quantized the quality scores using Quartz. Second, we extracted the quality scores from the resulting FASTQ file. Third, and finally, as recommended by the Quartz authors, we applied bzip2 on the modified quality scores.

In the case of QVZ2 we used the version available with commit hash d5383c6[11]. In the case of the QVZ2 software, the actual compression is already included, in contrast to Crumble and Quartz. With QVZ2 we performed the compression for the target MSE distortions 1 ("QVZ2 T1"), 2 ("QVZ2 T2"), 4 ("QVZ2 T4"), and 8 ("QVZ2 T8").

We used DSRC 2 [RD14] to mimic the 8-level binning introduced by Illumina. The specific mapping performed by DSRC 2 might differ slightly from the actual Illumina binning, as the latter might depend on the actual sequencing device. We compress the binned quality scores with gzip.

In summary, we evaluate 12 tool configurations on 3 chromosomes from 3 items. We conduct this evaluation using 6 different pipeline configurations. Hence, per item, we obtain $12 \cdot 3 \cdot 6 = 216$ rates and F_1 score values.

---

7 These few cases are mainly pertaining to the compressors "Crumble -9" and "IL8B".

8 https://github.com/jkbonfield/crumble

9 https://github.com/yunwilliamyu/quartz

10 The sequence dictionary used here is "dec200.bin.sorted", downloaded from https://giant.csail.mit.edu/quartz/dec200.bin.sorted.gz.

11 https://github.com/mikelhernaez/qvz2

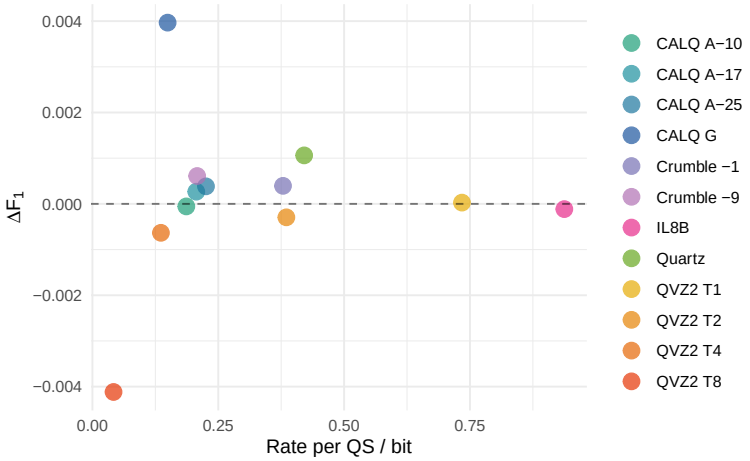


Figure 4.8: F_1 score difference versus rate per QS, item 1. Here, the general-purpose compressor gzip [Deu96] achieves a rate of 3.05 bit per QS.

4.5 RESULTS AND DISCUSSION

As stated above, using our experiment setup, we obtain 216 rates and F_1 score values per data item. For a concise[12] visualization of these results, we average the rates and F_1 score values over the chromosomes as well as over the variant calling pipeline configurations. Figure 4.8, Figure 4.9, and Figure 4.10 show the (averaged) F_1 score differences, with respect to the original quality scores, versus the (averaged) rates per QS for all tools.

Figure 4.8 shows the results for item 1. This item contains quality scores of the "Illumina 1.8+" type, exhibiting an alphabet size of 42 (see Table 4.1). As an anchor, here, the general-purpose compressor gzip [Deu96] achieves a rate of 3.05 bit per QS.

First, we investigate the results achieved by the Illumina 8-level binning (IL8B). This binning allows for more efficient compression, achieving a rate of 0.94 bit per QS. The impact on the F_1 score difference (-0.00011) is negligible.

Next, we investigate the results achieved by QVZ2. The quantization of quality scores in QVZ2 is controlled by a target MSE distortion. Larger target MSE distortions allow for a more coarse quantization and therefore

12 In addition to the concise visualization presented here we provide an extended set of figures in the appendix, where we average the rates and F_1 score values only over the chromosomes.

lead to lower rates. This observation is supported by the results shown in Figure 4.8. A target MSE distortion of 1 (QVZ2 T1) has a negligible impact on the F_1 score difference, and it leads to a rate of 0.73 bit per QS. Higher target MSE distortions (2, 4, and 8, respectively) impact the F_1 score difference (-0.00029, -0.00063, and -0.00412, respectively), but also allow for a more efficient compression (0.39 bit per QS, 0.14 bit per QS, and 0.04 bit per QS, respectively).

We can observe a similar behavior for our activity-based posterior model. Recall that we evaluated the activity-based posterior model for three different filtering configurations, i.e., we filter the normalized activity profile using $\sigma = 10$ (CALQ A-10), $\sigma = 17$ (CALQ A-17), and $\sigma = 25$ (CALQ A-25). Here, a higher value of σ leads to larger regions that are getting quantized with a higher resolution. Hence, we expect higher rates with growing σ. At the same time the hope is that we achieve a beneficial trade-off between rate and F_1 score difference. The results shown in Figure 4.8 corroborate our expectations. CALQ A-10 achieves a rate of 0.19 bit per QS, while at the same time exhibiting a minimal impact on the F_1 score difference (-0.00005). Filtering the normalized activity profile more cautiously (CALQ A-17 and CALQ A-25, respectively) leads to a slight increase in the rates (0.20 bit per QS and 0.23 bit per QS, respectively) as well as in the F_1 score difference (0.00027 and 0.00038, respectively). However, here we observe a *positive* F_1 score difference, i.e., the quantized quality scores lead to more accurate variant calls. Hence, we conclude that our quantization approach removes noise from the original quality scores. We also can conclude that the activity-based posterior model behaves as expected with regard to the chosen distortion metric, i.e., the F_1 score difference.

Next, we investigate the results obtained by our genotype likelihood model (CALQ G). With respect to the activity-based posterior model, we expect a lower rate for the genotype likelihood model, since in the latter only singular loci are quantized with higher resolutions, compared to entire regions (whose sizes depend on σ) in the former. The results shown in Figure 4.8 support our expectation. CALQ G achieves an average rate of 0.15 bit per QS. However, unexpectedly, the genotype likelihood model achieves an average F_1 score difference of 0.00397, which is superior compared to the activity-based posterior model. This observation is not further supported by the results that we obtained on items 11 and 12. Hence, we must conclude that the genotype likelihood model is unexpectedly overfitting on the specific data from item 1.

The first Crumble mode (Crumble -1) achieves a rate of 0.38 bit per QS at an F_1 score difference of 0.00039. Similar to our models, this tool also seems to be able to reduce the noise that is present in the quality

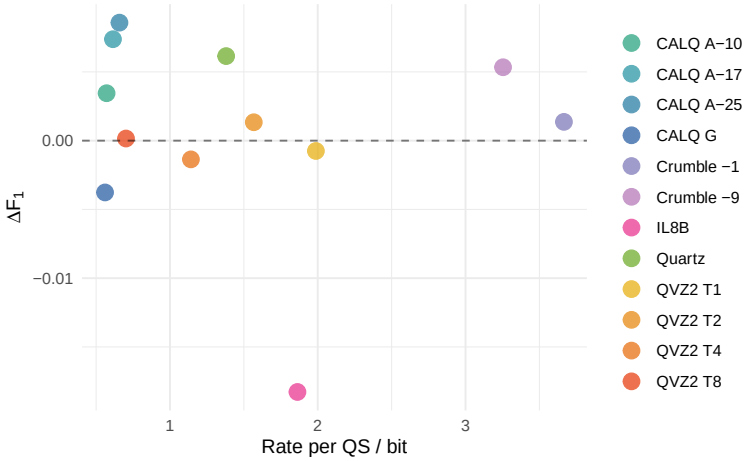


Figure 4.9: F_1 score difference versus rate per QS, item 11. Here, the general-purpose compressor gzip [Deu96] achieves a rate of 4.04 bit per QS.

scores. The second Crumble mode (Crumble -9), using a more aggressive binning strategy, achieves (as expected) a lower rate of 0.21 bit per QS at an F_1 score difference of 0.00061.

Similar to our models, and to Crumble, Quartz seems also to be able to remove noise from the data: it achieves a rate of 0.42 bit per QS at an F_1 score difference of 0.00106.

Figure 4.9 shows the results for item 11. This item contains quality scores of the "Sanger" type, exhibiting an alphabet size of 41 (see Table 4.1). As an anchor, here, the general-purpose compressor gzip [Deu96] achieves a rate of 4.04 bit per QS.

First, we investigate the results achieved by the Illumina 8-level binning (IL8B). As expected, it allows for more efficient compression, achieving a rate of 1.86 bit per QS. However, the impact on the F_1 score difference (-0.01830) is not negligible in this case, i.e., with regard to "Sanger" quality scores. This results was somehow expected, since the Illumina 8-level binning is tailored to Illumina data.

Next, we investigate the results achieved by QVZ2. Also, here, compared to item 1, higher target MSE distortions allow for a more efficient compression. However, in contrast to the results for item 1, higher target MSE distortions seem to not retainably affect the variant calling accuracy.

In the case of our activity-based posterior model we expect higher rates and more favorable F_1 score differences with growing σ. The results

shown in Figure 4.9 fully support our expectations. CALQ A-10, CALQ A-17, and CALQ A-25 achieve rates of 0.57 bit per QS, 0.61 bit per QS, and 0.66 bit per QS, respectively. At the same time we can observe growing F_1 score differences (0.00345, 0.00738, and 0.00858, respectively). As in the case of item 1, we observe positive F_1 score differences. Hence, we can take the same conclusion that our quantization approach removes noise from the original quality scores. Also, importantly, we can again conclude that the activity-based posterior model behaves as expected with regard to the chosen distortion metric, i.e., the F_1 score difference.

Next, we investigate the results obtained by our genotype likelihood model (CALQ G). Again, with respect to the activity-based posterior model, we expect a lower rate for the genotype likelihood model. Also, we expect a degradation in variant calling accuracy. The results shown in Figure 4.9 support our expectations. CALQ G achieves an average rate of 0.56 bit per QS at an F_1 score difference of -0.00377.

On item 11, Crumble achieves qualitatively similar results as on item 1. The first Crumble mode (Crumble -1) achieves a rate of 3.67 bit per QS at an F_1 score difference of 0.00137. Again, Crumble appears to be capable of reducing noise that is present in the quality scores. The second Crumble mode (Crumble -9), using a more aggressive binning strategy, achieves (as expected) a lower rate of 3.25 bit per QS at an F_1 score difference of 0.00534.

Similar to the results for item 1, and similar to our models as well as to Crumble, Quartz again seems to be able to remove noise from the data: it achieves a rate of 1.38 bit per QS at an F_1 score difference of 0.00615.

Figure 4.10 shows the results for item 12. This item contains quality scores of the "Illumina Reduced" type, i.e., Illumina's 8-level binning is applied as default. As an anchor, here, the general-purpose compressor gzip [Deu96] achieves a rate of 1.67 bit per QS.

First, we investigate the results achieved by QVZ2. Similar to the results obtained for item 1 and item 11, higher target MSE distortions allow for a more efficient compression. Similar to the results for item 11, and in contrast to the results for item 1, higher target MSE distortions seem to not retainably affect the variant calling accuracy.

Recall that in the case of our activity-based posterior model we expect higher rates and more favorable F_1 score differences with growing σ. Similar to the results for item 11, the results shown in Figure 4.10 fully support our expectations. CALQ A-10, CALQ A-17, and CALQ A-25 achieve rates of 0.51 bit per QS, 0.54 bit per QS, and 0.57 bit per QS, respectively. At the same time we can observe growing F_1 score differences (-0.00003, 0.00065, and 0.00068, respectively). Similar to the results for the other two items, we observe positive F_1 score differences. Hence, we

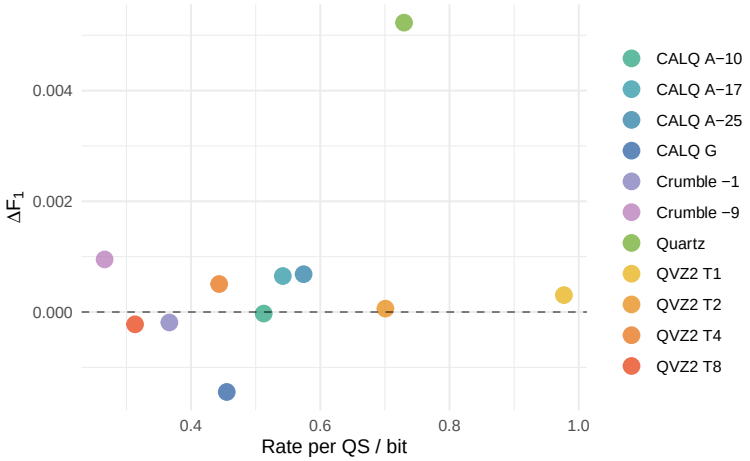


Figure 4.10: F_1 score difference versus rate per QS, item 12. Here, the general-purpose compressor gzip [Deu96] achieves a rate of 1.67 bit per QS. Note that the quality scores were generated with an Illumina HiSeq X Ten instrument (see Table 4.1). Hence, Illumina's 8-binning is applied as default. We therefore do not include the IL8B results in the plot.

can finally conclude that our quantization approach removes noise from the original quality scores. Moreover, we can finally conclude that the activity-based posterior model behaves as expected with regard to the chosen distortion metric, i.e., the F_1 score difference.

With respect to the activity-based posterior model, in the case of the genotype likelihood model (CALQ G), again, we expect a lower rate for the genotype likelihood model, and a degradation in variant calling accuracy. The results shown in Figure 4.10 support our expectations. CALQ G achieves an average rate of 0.46 bit per QS at an F_1 score difference of -0.00144.

Crumble achieves qualitatively similar results as on the other two items. The first Crumble mode (Crumble -1) achieves a rate of 0.37 bit per QS at an F_1 score difference of -0.00019. The second Crumble mode (Crumble -9), using a more aggressive binning strategy, achieves (as expected) a lower rate of 0.27 bit per QS at an F_1 score difference of 0.00095.

Quartz again seems to be able to remove noise from the data: it achieves a rate of 0.73 bit per QS at an F_1 score difference of 0.00523. This F_1 score difference is a positive outlier, compared to the results for the other tools. We hence conclude that the Quartz model is overfitting on the "Illumina Reduced" quality scores.

4.6 CONCLUSION

In this chapter, we investigated the compression of quality scores. We reviewed the state of the art, and we introduced our contribution CALQ. Besides the actual quality score quantization and entropy coding, CALQ consists of two proxy models, i.e., models that do not serve the purpose of redundancy reduction but that serve the purpose of controlling other parts of the compression process, in this case quantization.

The first proxy model is the genotype likelihood model, which infers the likelihood distribution of the genotype for each locus from the observable data, i.e., the read and alignment information, using a statistical model. The likelihood distribution of the genotype is further used to compute a "genotype uncertainty profile" over the loci, which is finally used to compute a quantizer index for each locus. The second proxy model is the activity-based posterior model, which builds on the genotype likelihood model. It contains a more realistic prior to infer the posterior distribution of the genotype for each locus. Note that we make use of an external reference sequence here. Also, further systematic assumptions are taken into account, to eventually compute an "activity profile" over the loci, which is also finally used to compute a quantizer index for each locus.

Each quantizer index (inferred using either model) identifies a specific precomputed quantizer to be used for the quantization of all quality scores at the current locus. Finally, the quality scores are quantized accordingly, and the quantizer indices as well as the quantized quality scores are encoded using entropy encoders. This way, high compression is achieved, while at the same time only a negligible impact on downstream analyses is observed.

The core of CALQ is the concept of locus-wise quantizer selection. This concept can be captured in a relatively simple decoding process syntax, where the specific model to be used for the inference of the quantizer indices is left open to the encoder design. Hence, the CALQ technology is a perfect fit for the incorporation into a compression standard, i.e., a decoding specification. Consequently, the CALQ technology was submitted to the standardization process of the ISO/IEC 23092 series, and it is now integrated in its part 2. In this chapter, we illustrated this integration in full detail.

In the evaluation of CALQ, it is of primary importance to assess the effect of the applied quantization on downstream applications. We elected variant calling as a representative downstream application here because it is critical to clinical decision making and therefore widely used. Building on previous works, we conceived an exhaustive experiment setup

consisting of different variant calling pipelines in various configurations. For CALQ and its competitors, we run each pipeline configuration with the quantized quality scores, and obtain a set of variant calls. We conduct a binary classification of each variant call set with regard to a truth set. As a measure of variant calling accuracy, we compute the F_1 score difference with respect to the original data. The pair of achieved rate per quality score and F_1 score difference is used to evaluate CALQ and its competitors.

From our results we can most importantly conclude that our models behave as expected with regard to rate per quality score and F_1 score difference. The predictable behavior of our models is in contrast to all competitors.

In summary, with CALQ, the quality scores can be compressed to an average rate of 0.39–0.49 bit per QS at an average F_1 score difference of -0.00041–0.00322 (CALQ G–CALQ A-25). For the application of quality score quantization in production scenarios, is it of high importance to ensure that the call set does not degrade under the influence of quantization. Our recommendation for such scenarios would be the use of CALQ A-17, which, in our experiments, leads exclusively to positive F_1 score differences at an average rate of 0.45 bit per QS.

A few competitors achieve better rates than CALQ (QVZ2 T4 and T8 on item 1 and item 12, as well as Crumble -1 and -9 on item 12). However, the results that we obtained for these competitors are not data-independent. We can hence finally conclude that our contribution CALQ is the only method available that can provide the best, as well as most reliable and predictable, rate-distortion results for the compression of quality scores.

Table 4.1: Selected test data from the MPEG-G Genomic Information Database [ISO20]. The entropy was estimated by compressing the quality scores with gzip [Deu96].

Item	Experiment Type	Species	Sequencing Technology Specifications

1	WGS	*H. sapiens*	– Technology: Sequencing by synthesis
			– Instrument: Illumina HiSeq 2000
			– QS type: "Illumina 1.8+"
			  – ASCII range: 33–74, Phred range: 0–41, alphabet size: 42
			  – Estimated entropy: 3.05 bit per QS

11	WGS	*H. sapiens*	– Technology: Ion Torrent semiconductor sequencing
			– Instrument: Ion Torrent PGM
			– QS type: "Sanger"
			  – ASCII range: 33–73, Phred range: 0–40, alphabet size: 41
			  – Estimated entropy: 4.04 bit per QS

12	WGS	*H. sapiens*	– Technology: Sequencing by synthesis
			– Instrument: Illumina HiSeq X Ten
			– QS type: "Illumina Reduced"
			  – ASCII set: {35,40,44,55,60,65,70,75}, alphabet size: 8
			  – Estimated entropy: 1.67 bit per QS

5

ENTROPY CODING OF DNA SEQUENCING DATA

As detailed in Section 2.4, compression techniques can be divided into two stages: modeling and entropy coding. In the modeling stage, redundant information is described in the form of a mathematical model. The data is then usually expressed in terms of a difference—the residual—between the data and the model. In the entropy coding stage, the residual is compressed, by representing every residual element with as few bits as possible. As mentioned in Section 2.4, data may also contain information that is considered irrelevant. Irrelevant information can be reduced by employing various quantization techniques. From the point of view of entropy coding, however, quantized data can simply be regarded as specially modeled data.

In this chapter, we investigate entropy coding methods in the context of compression of DNA sequencing data. First, we review the state of the art in Section 5.1. Here, we also introduce our contribution GABAC [Par+19; Vog+20], the premier implementation of an entropy codec compliant to ISO/IEC 23092-2. In Section 5.2, we present the architecture of GABAC in full detail. In principle, GABAC is a general-purpose codec. However, here we lay the focus on the performance of GABAC in typical DNA sequencing data compression scenarios. Therefore, to simulate such a scenario, and to be able to analyze the performance of GABAC with respect to the state of the art in DNA sequencing data compression, we conceived a well-grounded experiment setup, which we lay out in Section 5.3. We present and discuss the results in Section 5.4. Finally, we conclude our findings in Section 5.5.

5.1 STATE OF THE ART AND CONTRIBUTION

In general, entropy coding methods can be grouped into different classes. Static entropy coding methods, such as Elias gamma coding [Eli75], Fibonacci coding, and Golomb coding [Gol66]—including Rice coding [RP71]—provide fixed, i.e., static, mappings of input symbols to

variable-length code words. Non-static entropy coding methods, such as Huffman coding [Huf52], also map input symbols to variable-length code words; however, here, the codes are computed using the statistical properties of the source. Arithmetic coding methods in turn, such as arithmetic coding [WNC87], range coding, and CABAC [MSW03], encode an entire message into a single interval which is represented by two numbers. Similar to arithmetic coding methods, the more recent family of ANS methods encode an entire message into a single number (not an interval).

In this chapter, we investigate entropy coding methods in the context of compression of DNA sequencing data. State-of-the-art DNA sequencing data compression frameworks[1] employ all mentioned forms of entropy coding: they utilize static and non-static entropy coding methods, as well as arithmetic coding and ANS methods. For example, the BAM format uses BGZF, a format built on top of the gzip file format [Deu96]. Here, the actual compression is based on the LZ77 algorithm [ZL77] and Huffman coding [Huf52]. The Quip [Jon+12] and DeeZ[2] [HNS14] frameworks, in turn, are based on arithmetic coding. The CRAM [Fri+11] framework uses a collection of entropy coding methods, i.e., a mixture of experts approach, comprising the general-purpose tools gzip, bzip2, and xz, as well as a range variant of the family of ANS methods.

In this chapter, with regard to the state of the art, we focus on the CRAM and DeeZ frameworks, the two best-performing frameworks in the literature [Num+16]. The compression process implemented in both frameworks can be divided into the two stages modeling and entropy coding. However, in the respective modeling stages of both frameworks, the data is not modeled in terms of a mathematical model. Rather, it is merely split into a set of homogeneous descriptor streams, where each descriptor stream contains one specific type of data, e.g., mapping positions, quality scores, or mismatch information.

In contrast to the aforementioned frameworks, ISO/IEC 23092-2 specifies a compression method built around CABAC [MSW03].

Our contribution, GABAC [Par+19; Vog+20], is the premier implementation of a codec compliant to ISO/IEC 23092-2. GABAC integrates established technologies, such as CABAC [MSW03], binarization schemes, and transformations, into a straightforward solution for the compression of DNA sequencing data. The development of GABAC overlapped temporally with the development of ISO/IEC 23092-2. Hence, both development processes mutually influenced each other, and the GABAC

1 We use the term "framework" to refer to the combination of a format (e.g., file format) and compression technology.

2 After it publication, the DeeZ software has been modified to also use an ANS method.

development consequently resulted in multiple NB comments that were incorporated in ISO/IEC 23092-2. Although GABAC is intrinsically connected to ISO/IEC 23092-2, in what follows we show that integrating GABAC into the CRAM framework can lead to significant improvements.

5.2 GABAC ARCHITECTURE

In principle, GABAC is a general-purpose codec, i.e., it may compress any input byte stream. However, here we lay the focus on the performance of GABAC in typical DNA sequencing data compression scenarios. Hence, we assume that GABAC processes descriptor streams, as for example provided by the CRAM and DeeZ frameworks, or by an ISO/IEC 23092-2 implementation. In what follows, we explain the GABAC design from the encoder point of view. Accordingly, Figure 5.1 illustrates the GABAC architecture with a block diagram of the GABAC encoder. Note that the block diagram does not include the (trivial) input parsing stage.

Given an input descriptor stream (as a byte stream), the compression process implemented by GABAC consists of a five-stage pipeline: input parsing, (optional) 3-step transformation, binarization, context selection, and CABAC. Strictly speaking, only the last three stages, binarization, context selection, and CABAC, comprise an actual entropy coding technique. (The input parsing stage is a necessary preprocessing step, and the 3-step transformation is a collection of general-purpose modeling techniques.)

In the input parsing stage, the input byte stream is parsed into a stream of symbols.

The symbol stream is next processed by the 3-step transformation stage which converts the symbol stream into one or more transformed sub-streams. In the first transformation step ("sequence transformation") the input symbols are processed using one of the following three transformations (or no transformation, i.e., pass-through): i) equality coding, in which a symbol is replaced by a flag indicating equality with its predecessor and, if necessary, a correction symbol; ii) match coding, a transformation similar to the LZ77 [ZL77] algorithm; and iii) run-length coding, in which a repeated symbol is replaced by the symbol itself and its run-length. In the second transformation step ("LUT transformation"), a LUT transformation may be applied, where symbols are replaced using a LUT. Finally, in the third transformation step ("differential transformation") differential coding may be applied.

After the 3-step transformation stage, in the binarization and context selection stages, a binarization algorithm (to convert each symbol into a set of bits, called bin string) is selected along with a context selection

algorithm for each transformed sub-stream. In the final stage, each bin is combined with a context and both are processed using CABAC.

As elaborated above, the development of GABAC overlapped temporally with the development of ISO/IEC 23092-2. Consequently, the encoder architectures differ in a few points. To illustrate this, Figure 5.2 shows the architecture of the transformations and entropy coding stages in ISO/IEC 23092-2.

A comparison of Figure 5.1 and Figure 5.2 shows that the terminology that has finally been adopted in ISO/IEC 23092-2 is different from the terminology used in GABAC. For example, "RL Coding" in GABAC terminology becomes "RLE Coding" in ISO/IEC 23092-2 terminology. Also, the figure comparison reveals that ISO/IEC 23092-2 provides an additional (sub-)sequence transformation, "Merge Coding". Moreover, ISO/IEC 23092-2 provides additional binarizations. Finally, there are three structural differences with respect to the general architecture. First, in ISO/IEC 23092-2, symbols may be split after the subsequence transformation (depicted as box "Subsymbol Split" in Figure 5.2). For example, a symbol may be split into a lower half comprising a number of least significant bits, and into an upper half comprising the remaining most significant bits. Subsymbols are processed separately and independently from each other after the split. Second, in ISO/IEC 23092-2 it is only possible to use either the "LUT Transform" or "Diff Coding", whereas in GABAC, both transformations might be applied. Third, according to ISO/IEC 23092-2, the sign of symbols is split off and processed separately (see dotted arrow "Sign Flag" in Figure 5.2). In summary, we regard these architectural differences as minor, as the general architecture is preserved.

The following sections detail every stage of the GABAC encoder separately.

5.2.1 *Input Parsing*

In the input parsing stage, the input byte stream is parsed into a stream of symbols. For example, a symbol might occupy two consecutive bytes. In the input parsing stage these two bytes are interpreted as one symbol. The resulting stream of symbols is then processed by the 3-step transformation stage that converts the symbol stream into one or more transformed sub-streams.

5.2.2 *3-Step Transformation*

The 3-step transformation stage consists of the following steps: sequence transformation (Section 5.2.2.1), LUT transformation (Section 5.2.2.2), and differential transformation (Section 5.2.2.3). The goal of these transformations is to facilitate more efficient compression when using CABAC. Hence, they can be regarded as a collection of general-purpose modeling techniques. To achieve the goal of facilitating more efficient compression, the 3-step transformation process converts the input symbol stream into transformed sub-streams that typically exhibit: i) reduced numbers of symbols, by exploiting redundancy; ii) smaller alphabets, by converting the symbol stream into typically more homogeneous transformed sub-streams; iii) adapted distributions, by replacing symbols, and as such allowing for bin strings that are shorter and/or offer a higher predictability; iv) or a combination of the aforementioned points.

An example of such a combination is match coding, followed by LUT transformation. Match coding will typically reduce the number of symbols (a set of multiple symbols can be replaced by two symbols). The output transformed sub-streams of match coding usually each contain rather homogeneous data which can be compressed more efficiently. Furthermore, when LUT transformations are applied to these transformed sub-streams, frequent symbols will be replaced by symbols that can be represented by shorter bin strings, hence limiting the number of values that have to be encoded by CABAC.

Finally, it should be noted that these transformations are optional. Hence, they can be omitted if no coding efficiency is gained when applying them.

5.2.2.1 *Sequence Transformation*

In the first transformation step (depicted as box "Sequence Transformation" in Figure 5.1), one of three sequence transformations may be applied. The sequence transformation can also be omitted. Each sequence transformation creates two or three transformed sub-streams. In the further steps of the GABAC encoding process, the transformed sub-streams are processed separately (and separate encoding configurations may be used per sub-stream). The three available sequence transformations are: equality coding, match coding, and run-length coding.

EQUALITY CODING The input symbol stream is transformed into two transformed sub-streams. For each input symbol one or two output symbols are generated. The first output symbol t_1^{EQ} represents an equality

flag, indicating whether the input symbol is equal to the previous input symbol ($t_1^{EQ} = 1$) or not ($t_1^{EQ} = 0$). In the case that the input symbol is different from the previous input symbol the transformation will generate a second output symbol t_2^{EQ} depending on the current input symbol s_n and its predecessor s_{n-1}:

$$t_2^{EQ} = \begin{cases} s_n - 1, & \text{if } s_n > s_{n-1}, \\ s_n, & \text{otherwise.} \end{cases}$$

MATCH CODING The input symbol stream is transformed into three transformed sub-streams. During match coding, a search window is maintained. For each new set of input symbols, the search window is scanned for a matching block of symbols. Each set of input symbols is then replaced by two output symbols t_{length}^{M} and t_{data}^{M}. These two output symbols are stored across three different transformed sub-streams. The first output symbol t_{length}^{M} represents the length of the match, where $t_{length}^{M} = 0$ means that no match has been found. This output symbol is appended to the first transformed sub-stream. The second output symbol t_{data}^{M} can contain one of two different types of information, depending on the value of t_{length}^{M}. If $t_{length}^{M} = 0$ (i.e., no match has been found), t_{data}^{M} contains a raw symbol and is appended to the second transformed sub-stream. If $t_{length}^{M} \neq 0$, then t_{data}^{M} contains a pointer to the position of the first symbol of the match. In this case, t_{data}^{M} is appended to the third transformed sub-stream.

RUN-LENGTH CODING The input symbol stream is transformed into two transformed sub-streams. For each input symbol $s_n \neq s_{n-1}$, two output symbols are generated. The first output symbol t_1^{RLE} represents the input symbol s_{n-1}. The second output symbol t_2^{RLE} represents the number of repetitions of t_1^{RLE} in retrospect until the occurrence of a different input symbol.

5.2.2.2 *LUT Transformation*

In the second transformation step (depicted as box "LUT Transformation" in Figure 5.1), a LUT transformation may be applied, where each input symbol is replaced by an output symbol from a LUT.

Depending on the properties of the input symbol stream the use of a LUT can be of particular importance. The reason lies in the fact that, eventually, each symbol value will be converted into a bin string in the binarization stage (see Section 5.2.3). For this conversion, a set of predefined binarization processes is available, that provide fixed mappings

of symbols to bin strings. In general, less frequent symbols should get associated to shorter bin strings and vice versa. To achieve this goal, symbols can be shuffled using a LUT.

To illustrate this, let us assume that no sequence transformation (see Section 5.2.2.1) and no differential transformation (see Section 5.2.2.3) is used. Let us further assume that the input symbols follow a geometric distribution. Hence, it is beneficial to use the EG binarization process. However, let us also assume—as it is usually the case—that small input symbols do not correspond to high symbol frequencies. Hence, we have to employ a LUT that "correctly" maps symbols to new values according to the symbol frequencies (i.e., the symbol with the highest frequency is mapped to the value zero, etc.).

5.2.2.3 *Differential Transformation*

In the third transformation step, (depicted as box "Differential Transformation" in Figure 5.1), a differential transformation may be applied, where each input symbol is replaced by the arithmetic difference to its predecessor.

5.2.3 *Binarization*

In the binarization stage (depicted as box "Binarization" in Figure 5.1), the value of each symbol of each transformed sub-stream is converted into a bin string. In GABAC, six[3] binarization processes are available for the mapping of symbols to bin strings: Binary (Section 5.2.3.1), TU (Section 5.2.3.2), EG (Section 5.2.3.3), SEG (Section 5.2.3.4), TEG (Section 5.2.3.5), and STEG (Section 5.2.3.6).

5.2.3.1 *Binary Binarization*

Each input value v is represented by its binary representation.

5.2.3.2 *Truncated Unary Binarization*

Each input value v is represented by v ones followed by a zero. If v is equal to a defined maximal value[4] the trailing zero is discarded.

3 In ISO/IEC 23092-2, four additional binarization processes are specified, which are not implemented in GABAC. We are not aware of any other encoder making use of these binarizations.

4 This maximal value is an encoding parameter. In Figure 5.1 it is carried by the "Binarization Parameter" control signal.

5.2.3.3 *Exponential Golomb Binarization*

Each input value v is represented by a prefix and a suffix. The suffix is equal to the binary representation of $v + 1$. The prefix is expressed by a sequence of zeros whose length is equal to the length of the suffix minus one.

5.2.3.4 *Signed Exponential Golomb Binarization*

Input values $v \leqslant 0$ are mapped onto $-2 \cdot v$. Input values $v > 0$ are mapped onto $2 \cdot v - 1$. The resulting value is then converted into a bin string using the EG binarization.

5.2.3.5 *Truncated Exponential Golomb Binarization*

Each input value v is represented by one or two bin strings. Input values smaller than a given value v_{max} are represented by bin strings as per TU binarization. Input values larger than or equal to v_{max} are represented by bin string as per TU binarization of v_{max}, combined with the bin string as per EG binarization of the input value minus v_{max}.

5.2.3.6 *Signed Truncated Exponential Golomb Binarization*

Each input value v is represented by the TEG representation of $|v|$, and by a sign bit if $v \neq 0$.

5.2.4 *Context Selection and CABAC*

In parallel with the binarization stage, a context selection stage is performed (depicted as box "Context Selection" in Figure 5.1). A context is a number representing quantized probabilities. More specifically, it is a number between 0 (representing a probability of 0) and 127 (representing a probability of 1). It is used to encode a specific bin with CABAC, depicted as box "CABAC" in Figure 5.1 (see Section 2.4.4.4).

In GABAC, contexts are grouped into context sets. Each context set holds the contexts required to encode a bin string (where a bin string corresponds to a symbol).

GABAC provides four different modes of context selection. In the first mode (bypass) no context selection is performed and all bins are processed using contexts that assume equiprobability. Here, contexts are not adapted. In the second mode (order-0 adaptive coding), there is one context set for each symbol. Here, the contexts are adapted. In the third (order-1 adaptive coding) and fourth mode (order-2 adaptive coding),

context sets are selected based on the previously encoded symbol and based on the two previous encoded symbols, respectively. Also, here, the contexts are adapted.

5.3 EXPERIMENT SETUP

In principle, GABAC is a general-purpose codec, i.e., it may compress any input byte stream. However, here we lay the focus on the performance of GABAC in typical DNA sequencing data compression scenarios. Therefore, to simulate such a scenario, and to be able to analyze the performance of GABAC with respect to the state of the art in DNA sequencing data compression, we conceived a well-grounded experiment setup.

We assume that GABAC processes descriptor streams, as for example provided by the CRAM and DeeZ frameworks, or by an ISO/IEC 23092-2 implementation. Hence, to analyze the performance of GABAC, we modified the CRAM and DeeZ frameworks in such a way that they output their internal data representations, such as mapping positions and read pairing information, to separate descriptor stream files. Then, we encode every descriptor stream file with the entropy coding methods used in the CRAM framework: the general-purpose codecs gzip, bzip2, and xz, as well as a range variant of the family of ANS methods (rANS). Of course, we also encode every descriptor stream file with the GABAC encoder. The set of entropy coding methods used in the CRAM framework can safely be regarded as the state of the art, as other tools use a subset of these methods or arithmetic coding, which can be regarded as equivalent to rANS. In this way, we can evaluate the performance of GABAC against the state of the art.

In Section 5.3.1, we briefly revisit the used entropy coding methods. In Section 5.3.2, we detail our methodology for the generation of the CRAM and DeeZ descriptor stream files.

5.3.1 *Entropy Coding Methods*

The performance of GABAC was compared to the entropy coding methods, referred to as codecs here for brevity, that are used in the CRAM framework: the general-purpose codecs gzip, bzip2, and xz, as well as rANS.

All experiments were performed on a computer equipped with two Intel Xeon E5-2650 v3 CPUs and 128 GiB of RAM, running Ubuntu 14.04. Execution times were monitored with GNU time 1.7.

bzip2[5] uses various entropy coding techniques. The ones characterizing bzip2 are the Burrows-Wheeler transform and Huffman coding [Huf52]. In our experiments we used the "-9" flag to favor compression over speed as much as possible.

For GABAC, as depicted in Figure 5.1, various parameters may be used to control the encoding. Hence, in our experiments, we first analyzed each input descriptor stream file to infer the best possible set of parameters by evaluating every possible parameter combination with respect to the achieved compression. Then, in a second step, we conducted the actual encoding.

Gzip[6] is utilizing the LZ77 [ZL77] algorithm as well as Huffman coding [Huf52] for compression. In our experiments we used the "-9" flag to favor compression over speed as much as possible.

rANS is an algorithm from the family of ANS methods. We use the rANS implementation from HTSlib[7]. More specifically, we used rANS in two modes: in the mode "rANS-0" it is assumed that one symbol occupies a single byte; in the mode "rANS-1" it is assumed that one symbol occupies two[8] bytes.

xz is a command line application from the XZ Utils[9]. It provides an implementation of LZMA, which is based on a variant of the LZ77 [ZL77] algorithm and range coding. In our experiments we used the "-9" flag to favor compression over speed as much as possible.

5.3.2 *Test Data*

To evaluate the performance of GABAC, we modified the CRAM and DeeZ frameworks to output their internal data representations, such as mapping positions and read pairing information, to distinct descriptor stream files.

In a first step, to generate input for our modified versions of CRAM and DeeZ, we selected initial data from the MPEG-G Genomic Information Database [ISO20]. Specifically, we selected files pertaining to item 1 and item 2 from the MPEG-G Genomic Information Database [ISO20]. Table 5.1 shows the details of the selected files. Both items contain deep WGS data for members of the CEPH family 1463.

---

5 `http://www.bzip.org`

6 `https://www.gzip.org`

7 `https://www.htslib.org`

8 Hence, in the mode rANS-1 it is expected that the joint entropy (see Equation 2.20) of consecutive bytes is approached, i.e., it is expected that the mode rANS-1 is as least as efficient as the mode rANS-0.

9 `https://tukaani.org/xz`

For item 1, we decided to use the file pairs ERR174310 through ERR174314, as shown in Table 5.1. Here, we concatenated all files ending in "_1.fastq", and all files ending in "_2.fastq", respectively. Subsequently, we aligned the data following the GATK Best Practices[10]. The following tool versions were used for the alignment: BWA 0.7.13 [LD09], GATK 4.0.8.1 [McK+10; DeP+11; VO20], Picard 2.18.14[11], and Samtools 1.3 (built with HTSlib 1.3) [Li+09]. As reference we used the file "human_g1k_v37.fasta" from the GATK resource bundle version 2.8[12]. The file "NA12878_S1.sam" from item 2 is already aligned to another reference (GRCh37, also known as hg19). Finally, we extracted chromosome 11 from both resulting files. The final BAM files occupy 6.9 GiB (item 1) and 4.2 GiB (item 2), respectively.

Next, to generate the descriptor stream files[13], we processed both BAM files with our modified versions of CRAM and DeeZ. The resulting set of descriptor stream files contained some empty files which we removed before further processing. Moreover, for the CRAM descriptor stream files, we omitted[14] all descriptor streams not listed in the CRAM specification[15]. In summary, this process yielded: 28 CRAM descriptor stream files pertaining to item 1, chromosome 11; 37 CRAM descriptor stream files pertaining to item 2, chromosome 11; 39 DeeZ descriptor stream files pertaining to item 1, chromosome 11; and 52 DeeZ descriptor stream files pertaining to item 2, chromosome 11.

From these 65 CRAM descriptor stream files and 91 DeeZ descriptor stream files we omitted 7 files and 20 files, respectively, because they were too small (i.e., they contained only a few bytes) to provide reasonable performance numbers. Finally, this process yielded a test data corpus consisting of 58 CRAM descriptor stream files and 71 DeeZ descriptor stream files. To reduce the total test data corpus footprint, all remaining descriptor stream files were limited to 200 MiB. As a result, 18 files were capped at 200 MiB. This approach allows the analysis of a richer test data corpus while maintaining a reliable representation of compression performance for each of the entropy coding methods compared. The total size of the resulting test data corpus, consisting of 129 files, is approximately 8.9 GiB.

10 `https://gatk.broadinstitute.org`

11 `https://broadinstitute.github.io/picard`

12 `ftp://ftp.broadinstitute.org/bundle/b37`

13 We refer the reader to [Vog+20] for details on the semantics of the CRAM and DeeZ descriptor stream files.

14 We omitted all files with the following file name extensions: ".cram_raw", ".DS_Cram_Info", ".DS_<3-letter code>", ".DS_<number>".

15 `https://samtools.github.io/hts-specs/CRAMv3.pdf`

5.4 RESULTS AND DISCUSSION

As elaborated in Section 5.3, when conceiving and designing our experiment setup, we laid the focus on the performance of GABAC in typical DNA sequencing data compression scenarios. Therefore, our experiment is built around a specifically designed test data corpus, which consists of 129 descriptor stream files, where each file is limited to 200 MiB. We regard the compression of this set of descriptor stream files as a typical DNA sequencing data compression scenario where the data is organized in the form of descriptor streams, under the assumption that the splitting into descriptor streams facilitates more efficient compression. Also, by limiting each descriptor stream file to 200 MiB, we include the concept of rather small access units (that may be decompressed independently[16]) in our experiment.

In such a typical DNA sequencing data compression scenario, the most important question is which entropy coding method is the best-performing for every descriptor stream. Hence, we compute, for each descriptor stream, the rankings of the different codecs. In Figure 5.3, Figure 5.4, and Figure 5.5 we present, for each codec, the ranks achieved for the different descriptor streams along the compression axis, along the compression speed axis, and along the decompression speed axis, respectively.

For Figure 5.3, Figure 5.4, and Figure 5.5 the following semantics apply.

- Dots were jittered for visual clarity.
- The x-axes show the item from which the descriptor streams were generated.
- The y-axes show the actual ranks. Because we evaluate six codecs (bzip2, GABAC, gzip, rANS-0, rANS-1, xz) ranks are between 1 and 6.
- Each dot depicts the rank that a codec achieved on one specific descriptor stream.
- The boxes visualize the following statistics: the horizontal black line denotes the median, the lower and upper hinges correspond to the first and third quartiles, and the whiskers extend to the smallest/largest values no further than 1.5 times the inter-quartile range from the hinges.

16 Independently decompressible access units facilitate non-sequential access to the compressed data.

- The dotted lines denote the mean compression rank for each tool, averaged over both test items.

To put the compression ranks presented in Figure 5.3 in perspective, rANS-0 provides a "minimum anchor" for compression. Namely, if descriptor streams were viewed as "one-byte-per-symbol streams", then it would be expected that rANS-0 yields bitstreams whose sizes approach the (unconditional) entropy of the descriptor streams. Hence, one would expect that all other (more sophisticated) codecs outperform rANS-0 in all cases. In general, this behavior can be observed. However, it can be observed from Figure 5.3 that rANS-0 ranks first twice. It also ranks second twice, and it ranks third multiple times. These ranks are achieved by rANS-0 only on a few special descriptor stream files that are either very small or that contain extremely homogeneous data (such as a stream of zeros). In such cases, the other codecs produce too much overhead and are therefore outperformed by rANS-0.

In general, as it can be observed from Figure 5.3, GABAC obtains the best average compression rank (dotted line). The second best average compression rank is achieved by xz. To get an indication of the spread between the ranks, we calculated the average compression ratios for the group of first-ranked codecs (4.54) and the group of last-ranked codecs (2.94).

In addition to the compression ranks, as presented in Figure 5.3, and to provide another viewing angle at the results, Table 5.2 shows the compression ratios, averaged over both sets of descriptor streams.

It can be observed from Table 5.2 that GABAC, achieving an average compression ratio of 4.12, outperforms bzip2 (4.04), gzip (3.83), rANS-0 (3.01), and rANS-1 (3.76); it is itself only outperformed by xz (4.32).

Regarding the compression and decompression ranks, shown in Figure 5.4 and Figure 5.5, respectively, it should be noted that they were calculated without the times used to find the best GABAC parameter combination[17]. As it can be observed from Figure 5.4, GABAC is faster than gzip and xz in terms of the mean compression speed rank. In terms of the mean decompression speed rank GABAC is slower than the other codecs. At this point it is important to note that GABAC is a more or less straightforward implementation of the concepts presented in Section 5.2. In particular, the implementation has not been optimized, neither in a general way (e.g., loop unrolling), nor in particular ways (e.g., regarding architecture-specific optimizations).

In addition to the compression and decompression speed ranks, as presented in Figure 5.4 and Figure 5.5, respectively, Table 5.3 shows

17 Between 4,000 and 8,000 parameter combinations were evaluated per descriptor stream. Note that the number of possible parameter combinations depends on the actual input data.

the compression and decompression speeds, averaged over both sets of descriptor streams. The compression speed is calculated as uncompressed size divided by compression time, and the decompression speed is calculated as compressed size divided by decompression time.

It can be observed from Table 5.3 that, in terms of average compression speed, GABAC, operating at an average compression speed of 13.3 MiB/s, outperforms gzip (10.2 MiB/s) and xz (2.3 Mib/s). It is itself only outperformed by bzip2 (19.1 MiB/s), rANS-0 (45.5 MiB/s), and rANS-1 (38.3 MiB/s). In terms of decompression speed, GABAC operates at an average decompression speed of 1.8 MiB/s, ranking last in comparison to its competitors. Nonetheless, in a typical DNA sequencing data compression scenario, data is processed in the form of atomic units. In such a scenario the compression and decompression processes would very likely be highly parallelized, and thus multiple codec instances would be executed at the same time.

To further dissect the extent to which existing solutions can benefit from the inclusion of GABAC, Table 5.4 shows the compressed sizes for different codec sets, selecting for each descriptor stream the codec that gives the best compression: the codec set used in CRAM, the codec set used in CRAM plus GABAC, and the codec set used in CRAM plus GABAC, with gzip and rANS-0 removed. Table 5.4 shows that adding GABAC to the set of CRAM codecs further improves compression in all cases. Removing gzip does not affect the compression. (Recall that Figure 5.3 shows that gzip is never ranked first.) Removing rANS-0 has only a minor effect (< 0.005%).

5.5 CONCLUSION

In this chapter, we investigated entropy coding methods in the context of compression of DNA sequencing data. We reviewed the state of the art, and we introduced our contribution GABAC, the premier implementation of an entropy codec compliant to ISO/IEC 23092-2. GABAC provides proven technologies, such as CABAC, binarization schemes, and transformations, into a straightforward solution for the compression of DNA sequencing data, as well as also other arbitrary data (i.e., GABAC can be regarded as a general-purpose codec). The development of GABAC overlapped temporally with the development of ISO/IEC 23092-2. Hence, both development processes influenced each other, and the GABAC development therefore resulted in multiple NB comments that were incorporated in ISO/IEC 23092-2.

For the evaluation of GABAC, we conceived a well-grounded experiment setup, in which we laid the focus on evaluating the performance in

a typical DNA sequencing data compression scenario. The experiment setup involves the generation of a test data corpus, consisting of a set of DNA sequencing data descriptor streams that were produced with modified versions of the compression frameworks CRAM and DeeZ. Moreover, the setup involves the comparison of GABAC to the competing codecs bzip2, gzip, rANS (in two modes) and xz. In summary, our experiment setup is designed to answer the question which entropy coding method is the best-performing per descriptor stream.

With regard to the experiment setup, a remaining topic is the optimization of the search of the best combination of GABAC parameters for a given input descriptor stream. In our experiments we used an exhaustive search. However, in a real-world scenario the search space could be limited to a set of reasonable combinations, and more efficient algorithms for the search of the best combination might be employed. Alternatively, parameter combinations could be preset for specific descriptor streams, under the assumption that the statistical properties of the descriptor streams do not change largely between different data sets.

Our results show that, in a per-descriptor-stream ranking of codecs, GABAC obtains the best compression ranks on average. In conclusion, we can state that the addition of GABAC as an entropy coding method to CRAM and DeeZ would be advantageous in terms of both attainable compression and speed.

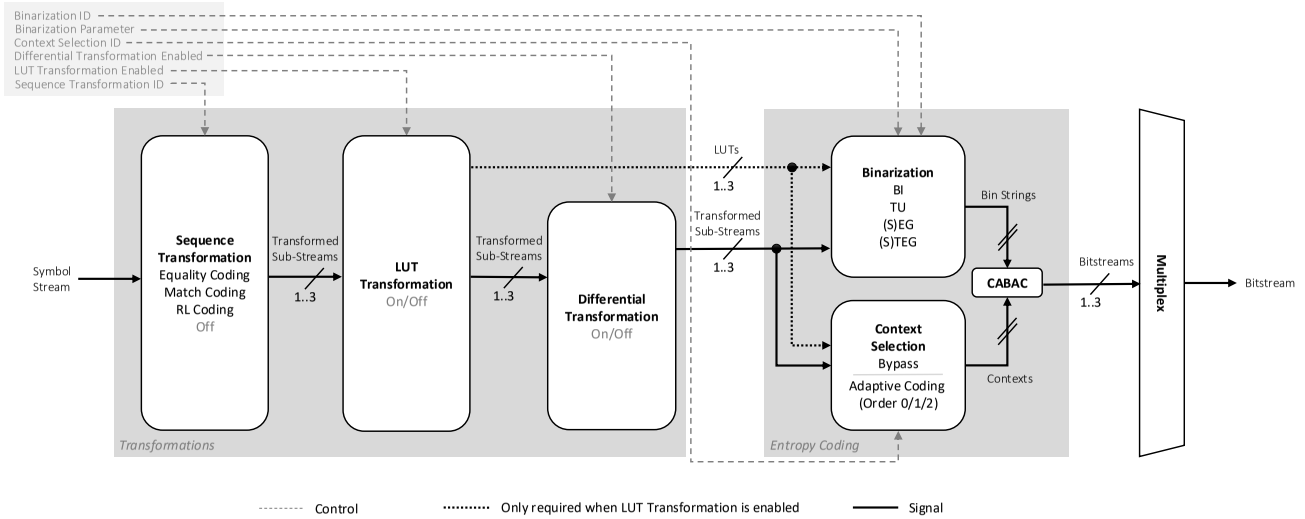


Figure 5.1: Block diagram of the GABAC encoder. Given an input descriptor stream, the compression process implemented by GABAC consists of a five-stage pipeline: input parsing, (optional) 3-step transformation, binarization, context selection, and CABAC. Here, the input parsing stage is not shown; it is assumed that the input is already a symbol stream. This input symbol stream is processed by the 3-step transformation stage that converts the symbol stream into one or more transformed sub-streams. The three transformation steps are visually grouped with a gray box. For each transformed sub-stream, a binarization algorithm (used for conversion of each symbol into a bin string) is picked, together with a context selection algorithm. Then, each bin is combined with a context and both are processed using CABAC. These last three stages, binarization, context selection, and CABAC, comprise the actual entropy coding part. Hence, they are also visually grouped with a gray box. The resulting bitstreams are multiplexed, yielding a single bitstream. Note that (optional) LUTs are also processed by the group of entropy coding stages. Finally, the top left of the figure shows the various parameters that may be used to control the encoding.

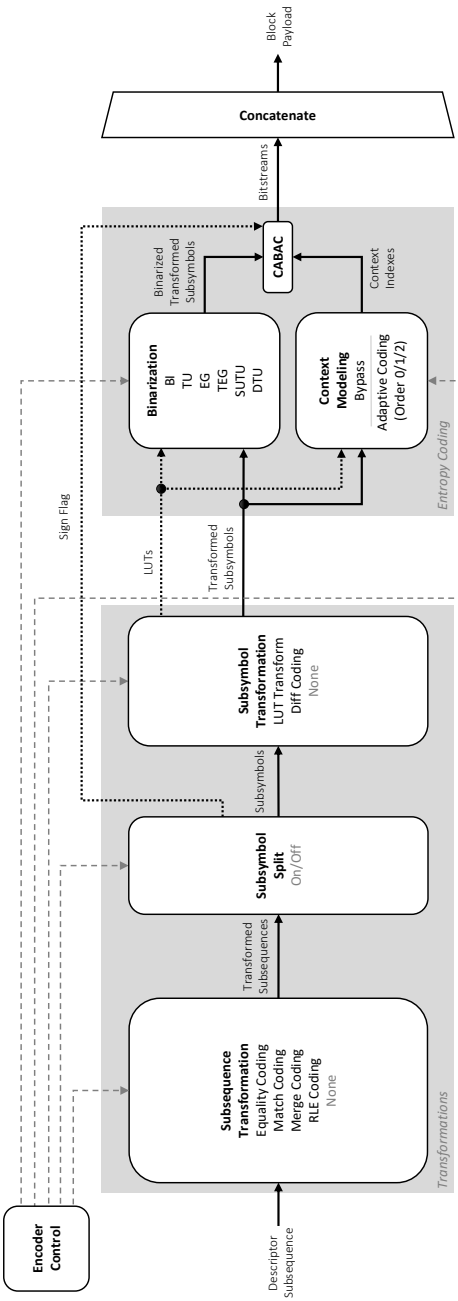


Figure 5.2: Block diagram of ISO/IEC 23092-2 transformations and entropy coding. The architecture exhibits only minor differences with respect to the architecture of the GABAC encoder (see Figure 5.1).

Table 5.1: Selected test data from the MPEG-G Genomic Information Database [ISO20].

Item	File Name(s)	Experiment Type	Species	Sequencing Technology
1	ERR174310_{1 | 2}.fastq ERR174311_{1 | 2}.fastq ERR174312_{1 | 2}.fastq ERR174313_{1 | 2}.fastq ERR174314_{1 | 2}.fastq	WGS	*H. sapiens*	Sequencing by synthesis
2	NA12878_S1.sam	WGS	*H. sapiens*	Sequencing by synthesis

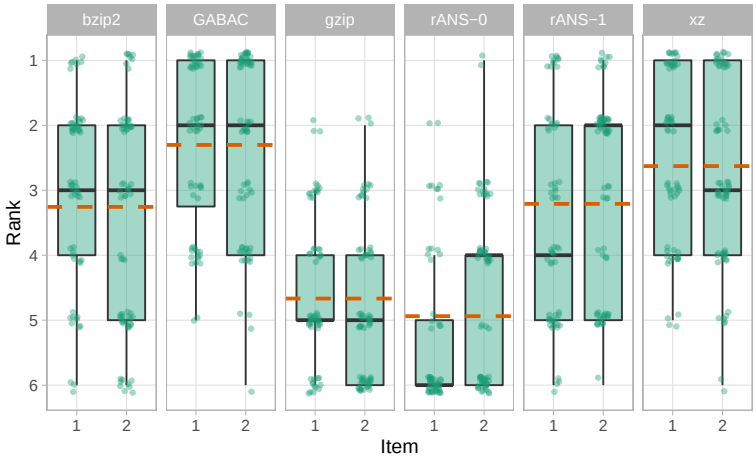


Figure 5.3: Compression ranks.

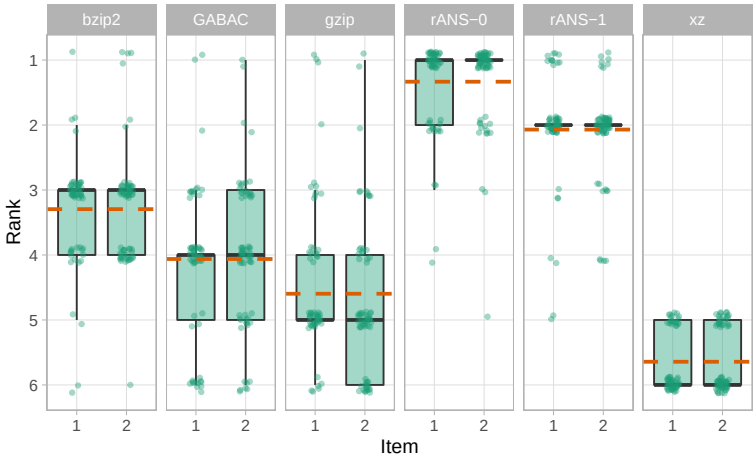


Figure 5.4: Compression speed ranks.

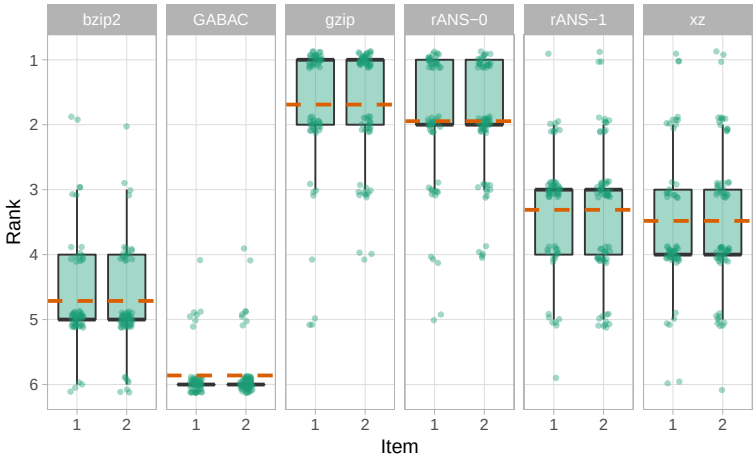


Figure 5.5: Decompression speed ranks.

Table 5.2: Compression ratios, averaged over both sets of descriptor streams.

Codec	∅ Compression Ratio
bzip2	4.04
GABAC	4.12
gzip	3.83
rANS-0	3.01
rANS-1	3.76
xz	4.32

Table 5.3: Compression and decompression speeds, averaged over both sets of descriptor streams.

Codec	∅ Compression Speed	∅ Decompression Speed
bzip2	19.1 MiB/s	3.7 MiB/s
GABAC	13.3 MiB/s	1.8 MiB/s
gzip	10.2 MiB/s	11.6 MiB/s
rANS-0	45.5 MiB/s	12.0 MiB/s
rANS-1	38.3 MiB/s	7.7 MiB/s
xz	2.3 MiB/s	5.1 MiB/s

Table 5.4: Compressed sizes (in bytes) for different codec sets applied to the CRAM and DeeZ descriptor streams. Best results in bold.

		Uncompressed	CRAM	CRAM +GABAC	CRAM +GABAC -gzip -rANS-0
CRAM streams	1	2,196,327,888	412,453,039	**409,337,321**	**409,337,321**
	2	1,991,360,268	406,796,124	**404,059,221**	404,080,825
DeeZ streams	1	2,838,530,295	538,155,557	**535,118,397**	**535,118,397**
	2	2,555,531,164	566,444,432	**564,839,381**	564,864,140

6

STANDARDS AND IMPLEMENTATIONS

The widely used FASTA format, the FASTQ format [Coc+10], and the SAM/BAM format [Li+09] were designed when sequencing data was scarce and precious. This becomes obvious by looking at their release dates—1985 (FASTA), around 2000 (FASTQ), and 2009 (SAM/BAM)—or by looking at their design. These formats are all (except the BAM format, which is the binary equivalent of the SAM format, in which the same data is stored in a compressed binary representation) human-readable textual file formats that are designed to be easily processable by command-line utilities such as awk, grep, and sed, which are featured on most Unix-like operating systems. This allows to draw conclusions about the environment in which these file formats were developed: sequencing data was generated in tiny amounts, compared to the amounts generated nowadays, and downstream processing was mainly conducted by specialized scientists who used handcrafted workflows tailored to specific use cases. In such settings, the properties of these file formats were highly beneficial.

However, the advent of HTS technologies prompted the need for more sophisticated ways of storing, transporting, and handling sequencing data. This demand catalyzed the creation of an abundance of specialized compressors for sequencing data (see [Num+16] and [Her+19]). Most of these compressors come with their own file format. In general, we refer to the combination of format and compression technology as framework. Most of these specialized frameworks are highly efficient, i.e., they provide competitive compression performance, primarily in terms of compression, but also regarding processing time and memory requirements. Some of them are well maintained[1], and some even come with a specification, which enables the creation of different implementations of a framework. However, what all of them lack is the combination of efficiency, a specification, and maintenance (in terms of integration of

1 Here, we refer to maintenance in terms of continuous adaption with regard to new technologies.

new technologies). Only this combination is what renders a framework truly usable.

In 2016, MPEG[2] together with WG 5 "Data processing and integration" of ISO TC 276 "Biotechnology" have started a standardization project with the hope that the availability of a framework for the representation of sequencing data that comes with a fully maintained specification will facilitate the creation of an ecosystem of efficient applications. This standardization project yielded the ISO/IEC 23092 series, which is also referred to as MPEG-G.

Besides MPEG and TC 276 there are also other organizations targeting the standardization of frameworks for the representation of sequencing data. Most notably, the industry alliance GA4GH maintains the specification of the CRAM framework [Fri+11], as well as other specifications for the representation and handling of sequencing data.

In this chapter, we first provide an overview of the sequencing data standardization landscape in Section 6.1. Next, we elaborate in detail on the ISO/IEC 23092 series[3] in Section 6.2. With regard to an introduction to the ISO/IEC 23092 series we also refer the reader to [Vog+21]. Most technologies presented in this dissertation were submitted to the standardization process of the ISO/IEC 23092 series, and they are now integrated in its part 2. Hence, in Section 6.2.2, we focus on part 2 (coding of genomic information). We provide more brief descriptions of part 1 (transport and storage of genomic information) in Section 6.2.1, part 3 (metadata and APIs) in Section 6.2.3, and parts 4 and 5 (reference software and conformance, respectively) in Section 6.2.4. Finally, in Section 6.3, we describe our software Genie, the first open-source implementation of the file format and compression technology specified in the ISO/IEC 23092 series.

2 MPEG is part of JTC 1 of ISO and IEC. Originally, it was a single WG (namely WG 11) organized within SC 29 of JTC 1. In 2020, SC 29 was subject to a restructuring process in which some subgroups of WG 11 became distinct WGs within SC 29. MPEG now comprises three AGs and seven WGs. The activities related to the ISO/IEC 23092 series are coordinated and conducted by WG 8 "MPEG Genomic Coding".

3 We use the term "ISO/IEC 23092 series" when referring to the entire standard series, including all its parts and editions. When referring to a particular part of the series, we usually use the notation "ISO/IEC 23092-X", where "X" is replaced by the respective part number. In the case that it is unambiguous that we refer to the ISO/IEC 23092 series, we sometimes use the shorter term "part X" (where "X" is again replaced by the respective part number) to refer to specific part. In the case that we need to refer to a specific edition of a part we use its full designation, e.g., ISO/IEC 23092-2:2020.

6.1 THE STANDARDIZATION LANDSCAPE

As most technologies presented in this dissertation were submitted to the standardization process of the ISO/IEC 23092 series, we focus in this chapter on the ISO/IEC 23092 series, with a particular emphasis on its part 2. However, ISO and IEC are not the only organizations standardizing frameworks for the representation of sequencing data.

Most notable, founded in 2013, GA4GH is an international consortium consisting of more than 600 members. It focuses on the development of standards for storing, analyzing, and sharing genomic data. GA4GH is maintained by three host institutions: Wellcome Sanger Institute[4], Broad Institute[5], and OICR[6]. Among others, GA4GH maintains the specification of the CRAM framework [Fri+11], detailing storage and compression of sequencing data. Concerning its scope, the CRAM framework can be compared to ISO/IEC 23092-1 (transport and storage of genomic information) and ISO/IEC 23092-2 (coding of genomic information). Also, GA4GH is maintaining the htsget [Kel+19] specification, which is detailing a streaming protocol for sequencing data, and the refget [Yat+21] specification, which is detailing APIs for accessing reference sequences. Concerning the scope of the htsget and refget specifications, they can loosely be compared to ISO/IEC 23092-3 (metadata and APIs).

Another international consortium is the Pistoia Alliance, which consists of more than 100 members. It was founded in 2009 by AstraZeneca, GlaxoSmithKline, Novartis, and Pfizer. It includes members from the pharmaceutical industry as well as not-for-profit, academic, and government institutions, and individual members. The scope of the Pistoia Alliance is very broad; its mission is to lower "barriers to R&D innovation by providing a legal framework to enable straightforward and secure pre-competitive collaboration"[7] between its members. Concerning the representation of sequencing data, in 2012, the Pistoia Alliance promoted an initiative for the comparison of tools for sequencing data compression: the SequenceSqueeze contest [BM13].

Furthermore, the WHO Expert Committee on Biological Standardization has the task to "establish detailed recommendations and guidelines for the manufacturing, licensing, and control of blood products, cell regulators, vaccines and related in vitro diagnostic tests"[8]. The committee does not have specific activity in the field of standardization of sequencing data representation.

4 https://www.sanger.ac.uk
5 https://www.broadinstitute.org
6 https://oicr.on.ca
7 https://www.pistoiaalliance.org
8 https://www.who.int/biologicals/expert_committee

Also, within the ISO hierarchy there are several committees that have a scope that is by some means connected to the activities of MPEG and TC 276. Most prominent are TC 212 "Clinical laboratory testing and in vitro diagnostic test systems" and TC 215 "Health informatics", here in particular SC 1 "Genomics Informatics". However, although they exhibit, in general, connections to the activities of MPEG and TC 276, these TCs do not pursue standardization projects in the field of sequencing data representation.

6.2 MPEG-G: THE ISO/IEC 23092 SERIES

The initial step of the standardization process of the ISO/IEC 23092 series was to define a list of requirements, such as support for streaming of compressed data and support for data protection [ISO15b]. This was done in the time between the 109th MPEG meeting in July 2014 and the 113th MPEG meeting in October 2015, where a public call for evidence [ISO15a], subject to the collected requirements, was issued. The evaluation of the responses to the call for evidence yielded promising results, and hence, a public call for proposals [ISO16b] was issued at the 115th MPEG meeting in June 2016. At the 116th MPEG meeting in October 2016, 10 responses to the call for proposals were received [ISO16c].

The technologies in the responses to the call for proposals were evaluated by judging their requirements coverage as well as their compression performance, primarily in terms of the size of the compressed data representation [ISO16a]. In a further ranking of proposals the computational complexity was evaluated both theoretically and empirically (by evaluating the processing times and memory requirements on a test data set). The most valuable technologies were combined, yielding an integrated framework for the transport, storage, and coding of sequencing data. As a result, the ISO/IEC 23092 series supports new features associated with complex use cases, most of which are not supported by the FASTA, FASTQ, and SAM/BAM formats.

Regarding the coding of sequencing data, the combination of the most valuable technologies yielded a multiplicity of modeling and coding modes. These modes facilitate the exploitation of the variety of statistical properties of different sequencing data types. In the case of unaligned nucleotide sequences, for instance, a model may be used where predictions can be made with respect to common patterns built around clusters of nucleotide sequences. In the case of alignments, for instance, reference sequences may be used for predictive coding. In addition, a classification system enables the grouping of alignments according to their alignment results. Such classification might be beneficial in certain use cases, where

for instance only perfectly matching reads are required. Also, such classification is expected to group data together that is expected to exhibit similar statistical properties, thus potentially improving compression. The integration of the most valuable technologies was achieved through the definition of the concept of "descriptors", structured in the form of "descriptor streams". These descriptor streams represent the sequencing data in a form that is beneficial for achieving high compression.

In addition to a framework for the transport, storage, and coding of sequencing data, the ISO/IEC 23092 series also includes the means, in terms of a reference software and a specification for assessing conformance, to verify that implementations comply with the different parts of the series. More specifically, part 5, "Conformance", specifies a set of test procedures to verify that bitstreams[9] and decoders meet the requirements of parts 1 and 2. These essential procedures enable the development of independent, yet compatible, implementations.

In summary, the ISO/IEC 23092 series currently consists of five parts, ISO/IEC 23092-1 to -5.

Part 1, "Transport and Storage of Genomic Information", specifies how sequencing data may be structured as a transport stream for transmission on a telecommunication network, or as a file for storage. Its main element is the specification of a hierarchical arrangement of data structures that may contain logically organized sequencing data. The two top-level data structures are the dataset group and the dataset. Also, part 1 specifies the make-up of metadata that may be attached to each dataset group or dataset. Moreover, part 1 provides a mechanism enabling non-sequential access inside each dataset: the MIT. Datasets are composed of various access units, and an access unit constitutes the smallest data structure that can be decoded by a decoder which is compliant to part 2. Hence, the access unit is the data structure that represents the link between part 1 and part 2. Part 1 also includes a reference process for converting a transport stream to a file[10].

Part 2, "Coding of Genomic Information", specifies the data structures to represent compressed sequencing data, as well as the associated decoding process. It encompasses the representation of (unaligned) reads as well as alignments (i.e., aligned reads), including associated quality scores and read identifiers. Part 2 also includes the representation of reference sequences. It is important to note that only the decoding process is

9 In the context of the ISO/IEC 23092 series, a bitstream is an ISO/IEC 23092-1 file, an ISO/IEC 23092-1 transport stream, or a sequence of concatenated ISO/IEC 23092-2 data units.

10 The reverse process (to convert a file into a transport stream) can be unambiguously deducted from the reference process.

specified, while any encoding process can be used, as long as it produces a bitstream that is compliant with this part.

Part 3, "Metadata and Application Programming Interfaces", specifies how metadata is attached to data structures. Here, the term "metadata" must not be confused with quality scores and read identifiers, which are contained in access units as specified in part 2. In the context of part 3, metadata refers to information metadata, providing general information (such as the information specified in the EGA[11] or the SRA[12] [Lei+11]), and protection metadata, which is covering three main aspects of protection: encryption, privacy, and integrity. Other functionalities covered by part 3 include the association of alignment metadata to compressed content, mechanisms for backward compatibility with existing content in the SAM format, and APIs.

Part 4, "Reference Software", provides two main applications as support and guide for the implementation of parts 1 and 2: the part 1 decoder application, referred to as decapsulator, and the part 2 decoder application, referred to as decoder. The reference software is normative in the sense that any complying implementation of the decapsulation or decoding process that uses the same compressed bitstreams and the same output data structures must output the same data.

Part 5, "Conformance", provides the means to test and validate the correct implementation of parts 1 and 2 to ensure full interoperability among all systems.

6.2.1 *Transport and Storage of Genomic Information*

The ISO/IEC 23092 series specifies in its part 1 the transport format and file format for the representation of sequencing data that is compressed according to part 2.

More precisely, the format that is intended to be used for the transport of packetized data on a telecommunication network is referred to as transport format. The format used for storage on a physical medium is referred to as file format. The actual incarnation of sequencing data according to the transport format and file format is referred to as ISO/IEC 23092-1 transport stream and ISO/IEC 23092-1 file, respectively.

These two formats are loosely based on the ISO/IEC base media file format (ISO/IEC 14496-12), and they are fully reversible: an ISO/IEC 23092-1 file can be converted into an ISO/IEC 23092-1 transport stream, which, in turn, can be converted back into an ISO/IEC 23092-1 file. To illustrate this, the conversion from a transport stream into a file

11 https://ega-archive.org

12 https://trace.ncbi.nlm.nih.gov

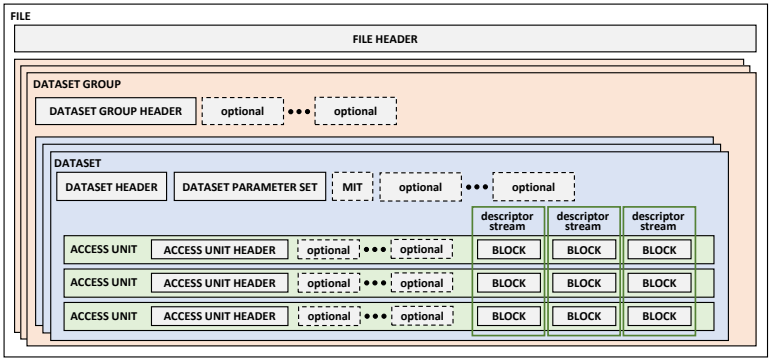


Figure 6.1: Key elements of the ISO/IEC 23092-1 file format. Multiple dataset groups contain multiple datasets of sequencing data. Each dataset is composed of access units. Each access unit contains blocks of coded data.

mainly involves the update of parameters on the dataset level and the compilation of the MIT (if present) from information residing in the access unit headers. Because of the similarity of transport format and file format, we focus in the following on the file format.

As shown in Figure 6.1, an ISO/IEC 23092-1 file is a hierarchical arrangement of data structures that contain logically organized sequencing data. At the top level, an ISO/IEC 23092-1 file is arranged into a file header and one or more dataset groups. Each dataset group contains a dataset group header as well as optional containers, and it encapsulates one or more datasets. Each dataset is composed by a dataset header, a dataset parameter set, and optional containers. It can also optionally contain a MIT, which facilitates non-sequential access inside each dataset. Each data set carries one or more access units. The access unit is the actual data structure that contains the compressed sequencing data. It constitutes the smallest data structure that can be decoded by a decoder which is compliant to part 2. Hence, the access unit data structure represents the link between part 1 and part 2. In addition to an access unit header and other optional containers, an access unit contains a collection of blocks, and each block contains a portion of a descriptor stream.

Each data structure in an ISO/IEC 23092-1 file is also associated to optional metadata ("information metadata") specified in part 3. This metadata provides general information about the data, such as the origin of the biological sample, a log of the operations performed on the data, and information associated to the preparation of the samples and the sequencing process. In addition, protection information ("protection

metadata"), which is also specified in part 3, can be associated to each data structure, providing the support for different selective protection approaches of the data.

The hierarchical design of an ISO/IEC 23092-1 file facilitates an abundance of use cases. For instance, the file can simply contain the data from a single sequencing run of a portion of a human chromosome. In another scenario, with regard to WGS experiments, an ISO/IEC 23092-1 file could be used to organize the storage of sequencing data for a trio of individuals (father, mother, child) as follows: there would be three different dataset groups, one for each individual in the trio. Each dataset group would contain datasets related to sequencing runs for the same individual, performed at different times. This example shows how ISO/IEC 23092-1 files can be used in a broad range of use cases, from long-term archiving to streaming scenarios.

6.2.2 *Coding of Genomic Information*

Part 2, "Coding of Genomic Information", specifies the data structures to represent compressed sequencing data. Most importantly, it also specifies the associated decoding process. Although part 2 specifies the decoding process, in this section, we present part 2 from the encoder perspective, because the employed technology is usually much easier to explain (and comprehend) when looking at it from the encoder side.

Figure 6.2 (page 129) shows the block diagram of the general ISO/IEC 23092-2 encoding process. The encoding process consists of three main stages: preprocessing, descriptor generation, as well as transformation and entropy coding. Each stage will be elaborated on separately in Sections 6.2.2.1, 6.2.2.2, and 6.2.2.3, respectively.

6.2.2.1 *Preprocessing*

Unaligned sequencing data consists of the actual nucleotide sequences, associated quality scores, and read identifiers. An incarnation of this triplet is referred to as an unaligned genomic record. An aligned genomic record contains alignment information and optional alignment metadata in addition. Prior to encoding, a part 2 encoder implementation is free to classify genomic records into six classes according to the result of the alignment of their nucleotide sequences against one or more reference sequences. Unaligned genomic records have a dedicated class (U). Aligned genomic records can be assigned to one of the other classes. All class semantics are summarized in Table 6.1.

Table 6.1: ISO/IEC 23092-2 data classes.

Class Name	Semantics
P	Reads perfectly matching to a reference sequence
N	Reads containing mismatches with respect to a reference sequence, which are unknown nucleotides only
M	Reads containing, with respect to a reference sequence, at least one substitution, and possibly unknown nucleotides, but no insertions, no deletions, and no clipped nucleotides
I	Reads containing, with respect to a reference sequence, at least one insertion, deletion or clipped nucleotide, and possibly unknown nucleotides or substitutions
HM	Half-mapped read pairs where only one read is mapped
U	Unmapped reads

After the classification, as the final step of the preprocessing stage, and before further processing, genomic records are split into their constituents: nucleotide sequences, quality scores and read identifiers, as well as alignment information and alignment metadata in the case of aligned data.

6.2.2.2 *Descriptor Generation*

The input of the descriptor generation stage consists of nucleotide sequences, quality scores and read identifiers, as well as alignment information and alignment metadata in the case of aligned genomic records. These different types of data are processed separately. In what follows, the encoding of nucleotide sequences (and alignment information), quality scores, and read identifiers is presented in more detail. Further processing of alignment metadata is specified in part 3, and it is out of scope at this point.

CODING MODES FOR NUCLEOTIDE SEQUENCES For the coding of nucleotide sequences (and alignment information, in the case of aligned data), an encoder is free to choose between four approaches:

- External reference: a reference sequence is available as an external resource (locally or remotely).
- Embedded reference: the reference sequence is embedded as a dataset (in the same file).
- Computed reference: a reference sequence is computed from the nucleotide sequences already processed.
- No reference: nucleotide sequences are, in principle, forwarded verbatim to the transformation and entropy coding stage.

Each of these approaches yields a set of descriptor streams which contain all information that is necessary to reconstruct the nucleotide sequences (and alignment information, in the case of aligned data). The goal in designing the descriptor streams was to ensure uncorrelation among them, if possible. This approach ensures maximum compression efficiency once the descriptor streams are fed through the transformation and entropy coding stage.

When using a computed reference, the encoder can choose between four reference computation algorithms:

- Reference transformation ("RefTransform"): An available external reference is modified before encoding nucleotide sequences.
- Read concatenation ("PushIn"): The reference is created by straightforward concatenation of already encoded nucleotide sequences.
- Local assembly: A local assembly of the underlying nucleotide sequence is built. This reference computation algorithm was contributed by us, and we detail it in Chapter 3.
- Global assembly: Common patterns that are shared among several nucleotide sequences are identified. The patterns are encoded only once along with the nucleotides specific to each nucleotide sequence.

The algorithms "PushIn" and "global assembly" can also be used to encode unaligned data.

The selection of encoding techniques depends on the specific application scenario requirements. From an application point of view, unaligned data can for example be encoded according to different approaches, depending on the actual scenario at hand. Here, we illustrate two such application scenarios: a "low latency" scenario, in which the "no reference" approach is used, and a "high compression" scenario, in which the "computed reference" approach with the "global assembly" reference computation algorithm is used.

- Low latency: In a streaming scenario, when low latency has higher priority than compression, a high-throughput compression approach is desirable. In such case the "no reference" approach is used: the descriptor generation stage is bypassed, and genomic record are directly forwarded to the transformation and entropy coding stage. This approach enables streaming scenarios, such as a scenario in which the data needs to be transmitted to a remote device "on-the-fly".

- High compression: High compression is achieved by taking advantage of the high redundancy in the sequencing data, e.g., by using the global assembly reference computation algorithm. This approach achieves high compression. However, it requires the availability of the entire data, as well as some preprocessing steps that may affect compression latency. It is suitable, for example, for long-term storage. Preprocessing technologies which are suited for this approach are for example ORCOM [GDR15], HARC [CTW18], FaStore [Rog+18], and SPRING [Cha+19].

Also, aligned data can be encoded according to the actual application scenario at hand. Here, we illustrate two such scenarios: a "clinical study" scenario, in which a reference-based approach (i.e, "external reference" or "embedded reference") is used, and a "reference-free" scenario, in which the "computed reference" approach with the "local assembly" reference computation algorithm is used.

- Clinical study: In this scenario, a multitude of human WGS experiments is performed. Here, it it beneficial to represent reads originating from different experiments by their differences with respect to a single set of reference sequences (i.e., one sequence per chromosome). In the case that the data of the sequencing of multiple genomes is stored in multiple datasets within a single ISO/IEC 23092-1 file, the reference sequences may be embedded as additional datasets within the same ISO/IEC 23092-1 file. This means that reference sequences can be shared among datasets. Also, external reference sequences can be used, where the actual access mechanisms are detailed in part 1.

- Reference-free: In this approach, reads are compressed without referring to any reference sequence by applying the local assembly reference computation algorithm. A local assembly of the underlying sequence is built, and reference-based compression with respect to the local assembly is performed. In this case, access to any refer-

ence sequences is not needed at neither the encoder nor the decoder side.

CODING MODES FOR QUALITY SCORES As shown in Chapter 4, quantization of quality scores can not only significantly reduce storage requirements, but also provide similar and sometimes even better performance in variant calling compared to the uncompressed data. Therefore, according to part 2, quality scores can be encoded either in a lossless manner or in a quantized manner. When encoding quality scores losslessly, several transformations can be applied (see Section 6.2.2.3). Quantization of quality scores, however, can lead to a dramatic increase in compression. To minimize any quantization effects, part 2 provides several mechanisms to allow a fine-grained selection of quantization schemes.

In the case of unaligned reads, an encoder is free to choose any beneficial scalar quantization scheme. This includes quantization schemes such as those presented in [GSR16; Och+13; Yu+15; Mal+15; HOW16]. The used representative values are signaled to the decoder by the means of a so-called codebook. The quantized quality scores are signaled to a decoder as indices into this codebook.

In the case of alignments, part 2 introduces an additional dimension to fine-tune the quantization of quality scores: codebooks can be selected per locus. As an illustrative example, an encoder could choose to select codebooks per locus using a simple genotyping model, such as in [VOH17], [VOH18], and [Och+19]. Before entropy coding, the indices are split into separate streams per codebook. Note that an additional stream for the locus-associated codebook identifiers is also required. Finally, an encoder is also allowed to tune the quantization by selecting different codebooks per class as well as per access unit.

CODING MODES FOR READ IDENTIFIERS Read identifiers are divided into a series of tokens which can be of three main types: strings, digits, and single characters. A read identifier is represented as a set of differences and matches with respect to one of the previously encoded read identifiers. This approach is not tailored to any specific implementation of a sequencing manufacturer and simply assumes that the structure of read identifiers is largely constant within the same sequencing run. This method (or variants of it) has been used in compressors such as Samcomp [BM13], DeeZ [HNS14], FaStore [Rog+18], and AliCo [Och+19].

6.2.2.3 *Transformation and Entropy Coding*

Storing different types of data in separate (homogeneous) descriptor streams allows for a significantly better compression efficiency as each stream is usually likely to contain stationary data.

To compress the heterogeneous set of descriptors, part 2 specifies the use of CABAC [MSW03], which is used in popular video coding standards such as AVC/H.264 [Wie+03] and HEVC/H.265 [Sul+12], as well as the sequencing data compressors AFRESh [Par+17] and AQUa [Par+18]. By selecting CABAC, the implementation of compliant codecs is simplified significantly, as a wide range of implementations, both in hardware and in software, is currently available.

Given an input descriptor stream[13], the compression process consists of two main stages: transformation and entropy coding. Each step of the transformation stage is optional (except for the sign extraction as part of the subsymbol transformation, in case that a transformed subsequence contains signed symbols). The transformation stage comprises the following pipeline: subsequence transformation, subsymbol split (including sign flag extraction), and subsymbol transformation. The (mandatory) entropy coding stage consists of binarization, context modeling, and CABAC. We presented our implementation of the transformation and entropy coding process—GABAC—in Chapter 5.

6.2.2.4 *Decoding Process*

In addition to syntax and semantics of the compressed sequencing data, part 2 also defines the decoding process.

The normative input of the decoding process is a concatenation of data structures called data units. Data units can be of three types. A "raw reference" data unit embodies the coded representation of one or more reference sequences. A data unit can also contain parameters to be used during the decoding process; it is then referred to as "parameter set". A data unit containing the coded representation of actual nucleotide sequences and associated read identifiers, quality scores, etc. is referred to as "access unit". The block diagram of the general ISO/IEC 23092-2 encoding process shown in Figure 6.2 depicts in its bottom right corner how these three types of data units are multiplexed into a single bitstream.

Raw references and parameter sets are used during the decoding process of access units, but do not produce output. Figure 6.2 shows analogously how raw references and parameter sets are fed into the encoder as supplementary signals. It is the decoding process of access

13 In part 2 a descriptor stream is referred to as "descriptor subsequence".

units that produces a normative output in the form of so-called ISO/IEC 23092-2 records.

6.2.3 *Metadata and Application Programming Interfaces*

Part 3, "Metadata and Application Programming Interfaces" has essentially two goals: to specify the syntax and semantics of the metadata that can be attached to data structures, and to specify an API that provides interoperability between applications that are built on top of normative decoders (see Section 6.2.4). Other functionalities covered by part 3 include the association of alignment metadata to compressed content and mechanisms for backward compatibility with existing content in the SAM format.

Two main types of metadata are specified: information metadata and protection metadata. Information metadata is related to metadata as specified in data repositories such as the EGA[14] or the SRA[15] [Lei+11]. It also includes metadata as stored in the header of SAM/BAM files. This facilitates interoperability when converting metadata to and from already existing databases. Note, that, to facilitate adaption to a multitude of use cases, part 3 also details a mechanism for the extension of the information metadata. Protection metadata is related to the protection technology applied to access units (and descriptor streams), datasets, or dataset groups. Part 3 covers three main aspects of protection: encryption, privacy, and integrity. For more information on the specifics of these two types of metadata we refer the reader to [Nar20].

Another important functionality provided in part 3 is the specification of an API enabling standardized access to the data structures of part 1, part 2, and part 3. API operations are applied to data structures that are arranged in a hierarchy. Hence, the hierarchy level defines the scope of an operation. The considered hierarchy levels are dataset group, dataset, and access unit. Note that the content may be protected, i.e., whenever the caller of API methods is not authorized to access the full content only the content for which the caller is authorized is returned.

6.2.4 *Reference Software and Conformance*

The ISO/IEC 23092 series includes as its part 4 a reference software to assess conformance to the requirements of parts 1 and 2. Also, in general, reference software is useful in aiding users of a standard (series) to estab-

14 https://ega-archive.org
15 https://trace.ncbi.nlm.nih.gov

lish and test conformance, and thus interoperability, and to demonstrate the capabilities of a standard (series). More specifically, the reference software comprises two main applications (which are written in the C programming language): the part 1 decoder application, referred to as decapsulator, and the part 2 decoder application, referred to as decoder. The decapsulator accepts bitstreams that are the result of an encapsulation according to part 1, and the decoder accepts bitstreams encoded according to part 2. Both applications are compliant implementations of parts 1 and 2, respectively: taking the same input and using the same output format the applications will output the same content. It should be emphasized that the reference software is not an optimized implementation. Complying implementations of part 1 or part 2 are not required nor expected to follow the algorithms and programming techniques used in the reference software. Hence, it should not be used as a benchmark of computational performance.

Part 5, "Conformance", specifies a set of test procedures to verify that bitstreams and decapsulators as well as decoders meet the requirements of parts 1 and 2. For this purpose, an exhaustive set of bitstreams was generated. Every implementation claiming conformance to either part 1 or part 2 will have to demonstrate the correct decoding or decapsulation, respectively, of the corresponding set of bitstreams.

6.3 AN OPEN-SOURCE MPEG-G CODEC

The main technologies presented in this dissertation—TSC (see Chapter 3) and CALQ (see Chapter 4)—were submitted to the standardization process of the ISO/IEC 23092 series, and they are now integrated in its part 2. Also, the development of GABAC (see Chapter 5) overlapped temporally with the development of the ISO/IEC 23092 series, and hence, both development processes influenced each other.

Consequently, we undertook the effort to join the implementations of TSC, CALQ, and GABAC. Eventually, this work resulted in our software Genie[16], the first open-source implementation of the file format and compression technology specified in the ISO/IEC 23092 series.

In Section 6.2.4 we mentioned that the ISO/IEC 23092 series includes as its part 4 a reference software. We want to clarify at this point that Genie is a software which is orthogonal to the reference software. More in detail, the reference software provides the implementation of the

16 https://github.com/mitogen/genie

decoding processes[17] that are specified in ISO/IEC 23092-1 and ISO/IEC 23092-2. Genie, in turn, provides a possible, yet not exhaustive or optimized, implementation of the encapsulation (ISO/IEC 23092-1) and encoding processes (ISO/IEC 23092-2) that produce bitstreams which can be decoded by the reference software. In addition to the encapsulation and encoding processes, Genie also includes implementations of the decapsulation and decoding processes; however, these (non-normative) implementations are distinct from those in the reference software.

17 The specification of the decoding processes (in contrast to the specification of the encoding processes) guarantees the interoperability of implementations of the ISO/IEC 23092 series, while leaving the encoding processes open to improvements.

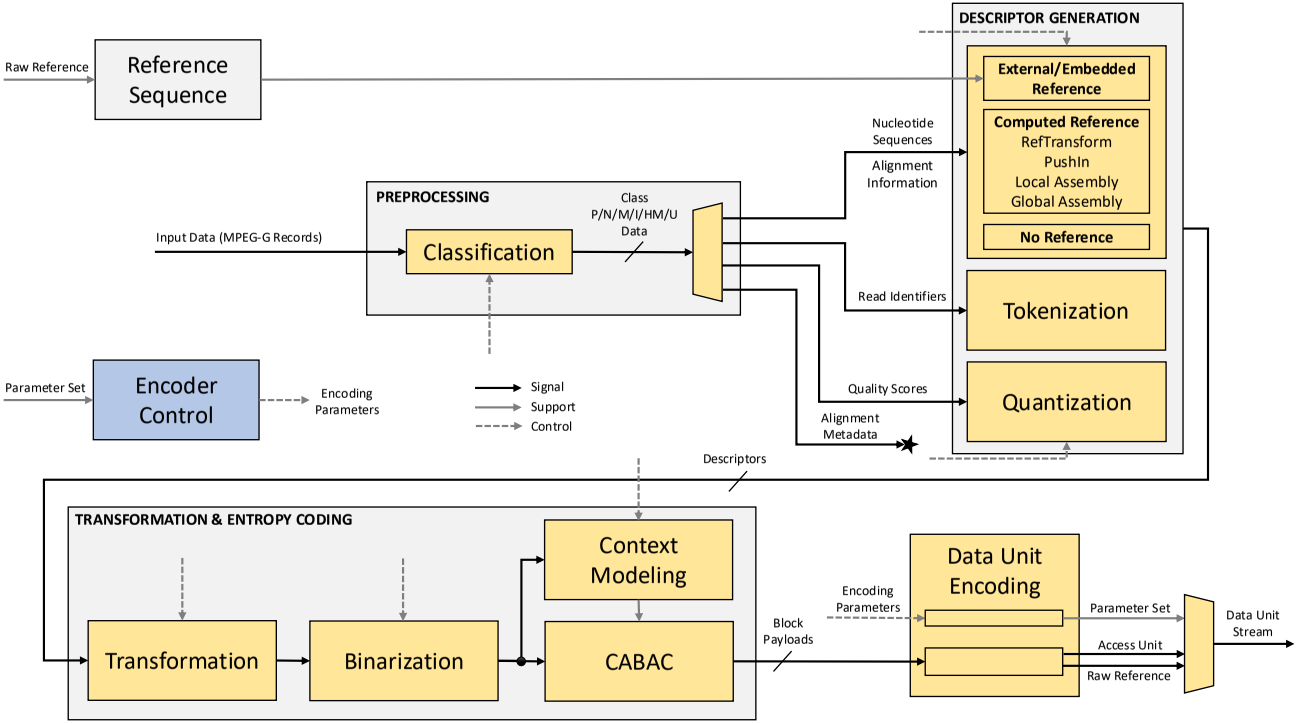


Figure 6.2: Block diagram of the general ISO/IEC 23092-2 encoding process. The encoding process consists of three main stages: preprocessing, descriptor generation, as well as transformation and entropy coding. Note that a parameter set can be re-used by feeding it back into the encoder control, and that a raw reference can also be re-used by initializing a reference sequence from it. Also note that alignment metadata may be attached to access units; however, this is specified in part 3 and hence not further detailed in the figure.

7

CONCLUSIONS

With the release of the latest generations of sequencing machines, the cost of sequencing a whole human genome has dropped to less than US$1,000. The potential applications in several fields—such as precision medicine and oncology—lead to the forecast that the amount of DNA sequencing data will soon surpass the volume of other types of data, such as video data. The costs related to storage, transmission, and processing of these large volumes of DNA sequencing data are already comparable to the sequencing costs. Therefore, the overarching objective of this dissertation is the development of data compression technologies that facilitate scalable DNA sequencing data storage, transmission, and processing.

In Chapter 1, we laid out the motivation for this dissertation in more detail, and we put our contributions in context to the state of the art. Also, we provided an outline of the dissertation.

In Chapter 2, we introduced the foundations that are necessary for the understanding of the methods that we present in this dissertation. These foundations span multiple disciplines that range from molecular biology over DNA sequencing and bioinformatics to data compression. With regard to data compression, we put a special emphasis on the concepts of modeling and coding, as we make extensive use of these concepts in our contributions.

In Chapter 3, we investigated the compression of aligned reads. We reviewed the state of the start, and, making use of the concept of modeling, introduced our contribution TSC for the modeling of aligned reads. In contrast to the state of the art, TSC is designed to exhibit an extremely low memory footprint, and to be able to operate without external reference sequences. TSC consists of two main elements. The first main element is the use of a sliding window to track recently encoded reads. The second main element is the use of the data tracked within the sliding window to infer a so-called local sliding consensus reference for the prediction of subsequent reads. These two elements have been adopted in ISO/IEC 23092-2, and we showed how these elements are integrated

in ISO/IEC 23092-2:2020. We coupled TSC with generic compression methods to evaluate its performance with respect to the state of the art. For our evaluation, we made use of a previously published benchmark suite, that we co-authored. Our results show that TSC provides the best trade-off of memory usage and achieved compressed size. Among the codecs that are specialized on the compression of aligned reads, TSC exhibits the lowest average memory consumption of 563 MiB. At the same time, TSC reaches an average compressed size of 83.05% with respect to the anchor BAM, where the other specialized non-reference-based codec only reaches 99.76%.

In Chapter 4, we investigated the compression of quality scores. We reviewed the state of the art, and, again making use of the concept of modeling, introduced our contribution CALQ for the quantization and compression of quality scores. Besides the actual quality score quantization and entropy coding, CALQ consists of two proxy models, i.e., models that do not serve the purpose of redundancy reduction, but that serve the purpose of controlling other parts of the compression process, in this case quantization. The first proxy model is the genotype likelihood model, which infers the likelihood distribution of the genotype for each locus from the observable data, i.e., the read and alignment information, using a statistical model. The second proxy model is the activity-based posterior model, which builds on the genotype likelihood model by adding a more realistic prior for the inference of the posterior distribution of the genotype for each locus, and by taking into account further systematic assumptions. Either proxy model is used to infer a specific precomputed quantizer to be used for the quantization of all quality scores at each locus. Finally, the quality scores are quantized accordingly, and the quantizer indices as well as the quantized quality scores are encoded using entropy encoders. This way, high compression is achieved, while at the same time only a negligible impact on downstream analyses is observed. This concept can be captured in a relatively simple decoding process syntax, where the specific model to be used for the inference of the quantizer indices is left open to the encoder design. Consequently, the CALQ technology was submitted to the standardization process of the ISO/IEC 23092 series, and it is now integrated in its part 2. Regarding the evaluation of CALQ, it is of primary importance to assess the effect of the applied quantization on downstream applications. We selected variant calling as a representative downstream application. As a measure of variant calling accuracy under the impact of quantized quality scores, we used the F_1 score difference with respect to the original data. Consequently, the pair of achieved rate per quality score and F_1 score difference is used to evaluate CALQ and its competitors. From our results

we can most importantly conclude that, in general, our models behave as expected with regard to rate per quality score and F_1 score difference, i.e., a decrease in the rate per quality score is associated to a decrease in the F_1 score difference. This predictable behavior of our models is in contrast to all competitors. In summary, with CALQ, using the different models, the quality scores can be compressed to an average rate of 0.39–0.49 bit per quality score at an average F_1 score difference of -0.00041–0.00322. In particular, we can recommend a particular parametrization for the activity-based posterior model, which, in our experiments, leads to exclusively positive F_1 score differences. This parametrization results in an average rate of 0.45 bit per quality score. This finding finally removes the major obstacle—i.e., a possible degradation in the quality of the call set—that hinders the wide-spread use of quality score quantization in production scenarios. In summary, we can conclude that our contribution CALQ provides the best as well as most reliable and predictable rate-distortion results for the compression of quality scores.

In Chapter 5, we investigated entropy coding methods in the context of compression of DNA sequencing data. We reviewed the state of the art, and we introduced our contribution GABAC, the premier implementation of an entropy codec compliant to ISO/IEC 23092-2. GABAC combines proven technologies, such as CABAC, binarization schemes, and transformations, into a straightforward solution for the compression of DNA sequencing data. The development of GABAC overlapped temporally with the development of ISO/IEC 23092-2. Hence, both development processes influenced each other, and GABAC was incorporated in ISO/IEC 23092-2. For the evaluation of GABAC, we conceived a well-grounded experiment setup, in which we laid the focus on evaluating the performance in a typical DNA sequencing data compression scenario. Our experiment setup is designed to answer the question which entropy coding method is the best-performing for every type of DNA sequencing data. Our results show that GABAC obtains the best compression ranks on average. We can also conclude that adding GABAC as entropy coding method to the compression frameworks CRAM and DeeZ would be beneficial both in terms of achievable compression and speed.

Finally, in Chapter 6, we provide an overview of the sequencing data standardization landscape, and we elaborate in detail on the ISO/IEC 23092 series. The main technologies presented in this dissertation—TSC, CALQ, and GABAC—are now integrated in its part 2. Hence, in our explanations of the ISO/IEC 23092 series, we put a special focus on its part 2. Finally, we describe our software Genie, the first open-source implementation of the file format and compression technology specified in the ISO/IEC 23092 series.

In summary, and in our view, the contributions presented in this dissertation constitute major building blocks that can help create an entire ecosystem of DNA sequencing data applications. To assist in achieving this goal and to facilitate future research, the implementations of our contributions are made available online to the scientific community[1]. Specifically, regarding the compression of aligned reads, we view our contribution TSC as one of the final methods in a line of methods that all explore the compression performance space of compression, computational complexity, and functionality, where TSC focuses on a beneficial trade-off between all three dimensions. Moreover, we regard our contribution for the compression of quality scores, CALQ, as a method that operates near the pareto frontier in the equilibrium of rate and distortion. To put this in context, historically, the most successful quantization methods that approach the pareto frontier have been tailored to their respective applications (e.g., by using psychoacoustic and psychovisual models in audio as well as image and video coding). CALQ, being tailored towards variant calling, is in line with these successful methods. As for entropy coding of DNA sequencing data, most entropy coding approaches have already been evaluated in the literature. Adding to this prior work, we apply CABAC in our contribution GABAC, therefore opening the door to hardware-based optimized entropy coding of DNA sequencing data. Further research in the area of entropy coding of DNA sequencing data may involve the application of the family of ANS methods, and work in this direction is already underway. Finally, we hope that our contributions will ultimately support the democratization of DNA sequencing data, and help to realize its as yet undiscovered potential.

1 The respective URLs can be found in our publications [VMO16; VOH18; Vog+20].

APPENDIX

COMPRESSION OF QUALITY SCORES—EXTENDED RESULTS

In the assessment of our contribution CALQ in Section 4.5, we evaluate 12 tool configurations on 3 chromosomes from 3 items using 6 different variant calling pipeline configurations. Hence, per item, we obtain $12 \cdot 3 \cdot 6 = 216$ rates and F_1 score values. We show a concise visualization of these results by averaging the rates and F_1 score values over the chromosomes as well as over the variant calling pipeline configurations (see Figure 4.8, Figure 4.9, and Figure 4.10).

Recall that we selected three different variant calling pipelines for our evaluation, and that we use the first pipeline in four different configurations; this yields six different pipeline configurations in total. The first pipeline is composed according to the GATK best practices workflow for germline short variant discovery [DeP+11]. Here, VQSR with four different levels of filtering—90%, 99%, 99.9%, and 100%—is used to filter the called variants to remove false positives (see Section 4.4 for a more detailed description of this process). The second pipeline is also composed according to the GATK best practices workflow, but using the more basic hard filtering (HF) instead of VQSR. The third pipeline involves the variant caller Platypus [Rim+14].

In addition to the concise visualization presented in Section 4.5 we here provide an extended set of figures by averaging the rates and F_1 score values only over the chromosomes.

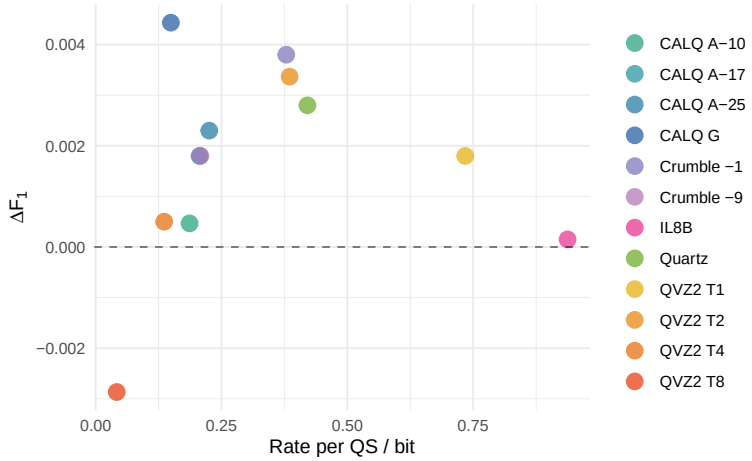


Figure A.1: F_1 score difference versus rate per QS, item 1, GATK VQSR 90.0%.

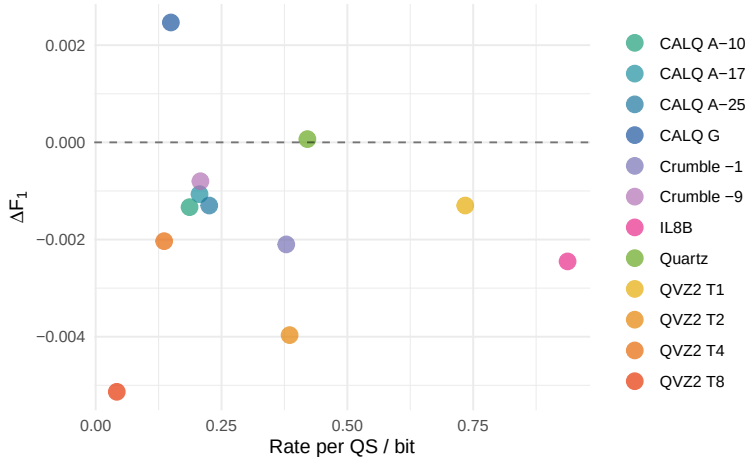


Figure A.2: F_1 score difference versus rate per QS, item 1, GATK VQSR 99.0%.

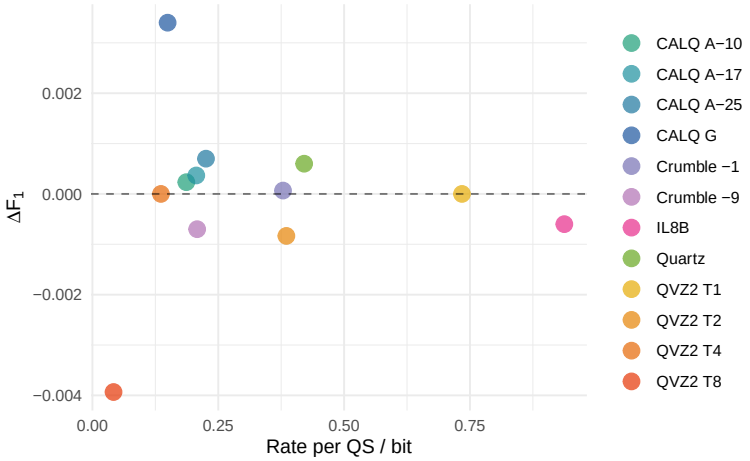


Figure A.3: F_1 score difference versus rate per QS, item 1, GATK VQSR 99.9%.

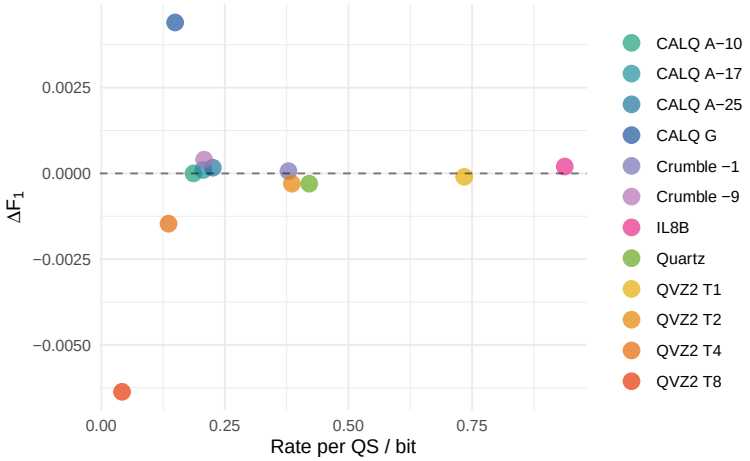


Figure A.4: F_1 score difference versus rate per QS, item 1, GATK VQSR 100.0%.

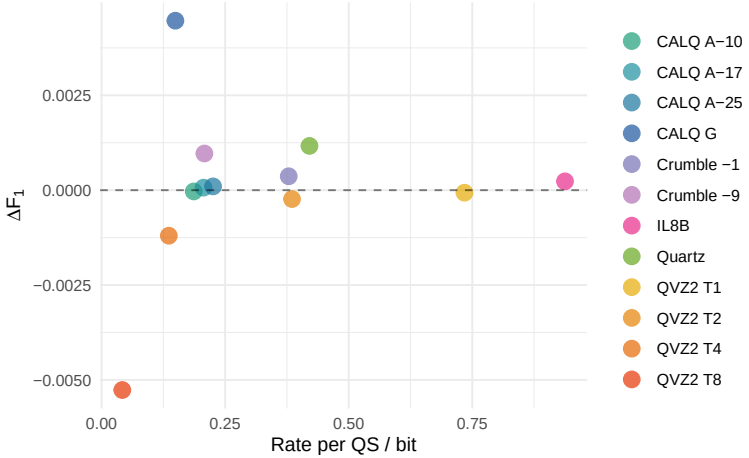


Figure A.5: F_1 score difference versus rate per QS, item 1, GATK HF.

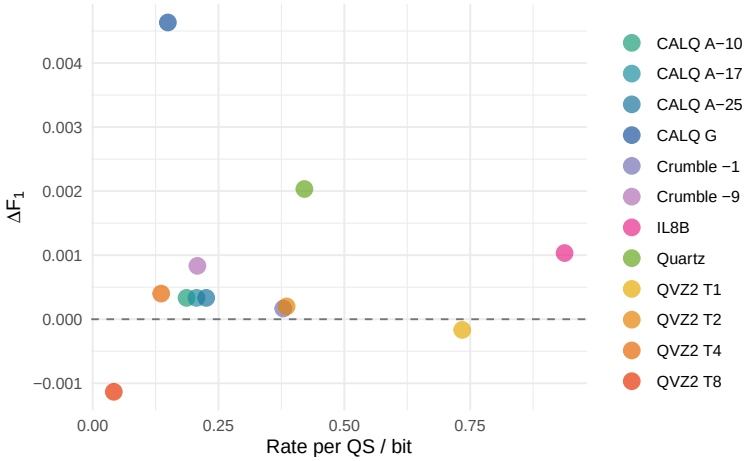


Figure A.6: F_1 score difference versus rate per QS, item 1, Platypus.

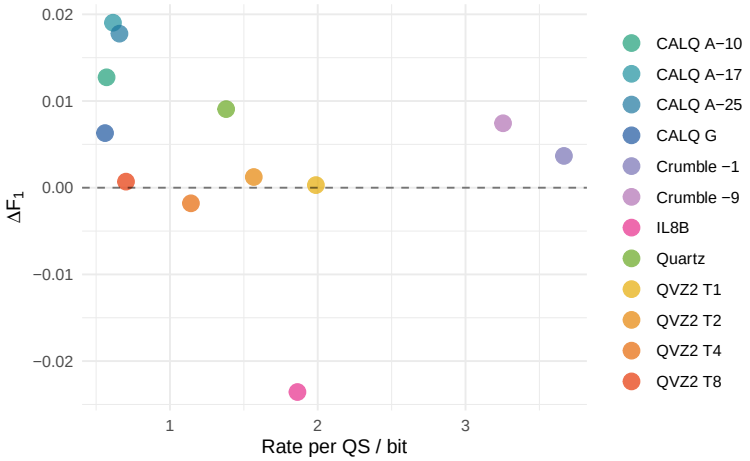


Figure A.7: F_1 score difference versus rate per QS, item 11, GATK VQSR 90.0%.

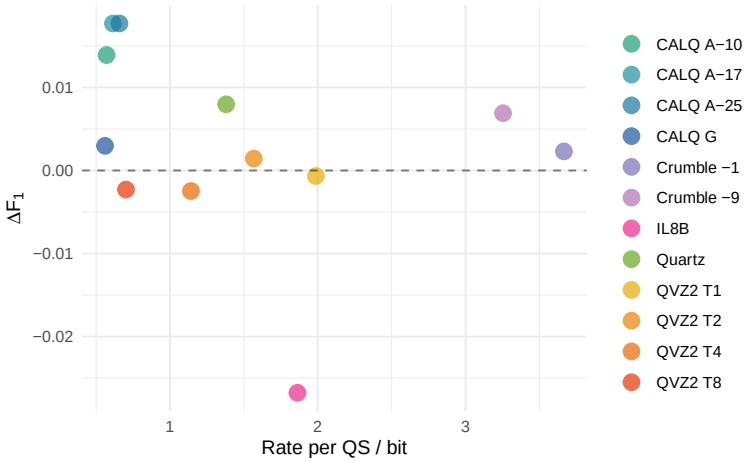


Figure A.8: F_1 score difference versus rate per QS, item 11, GATK VQSR 99.0%.

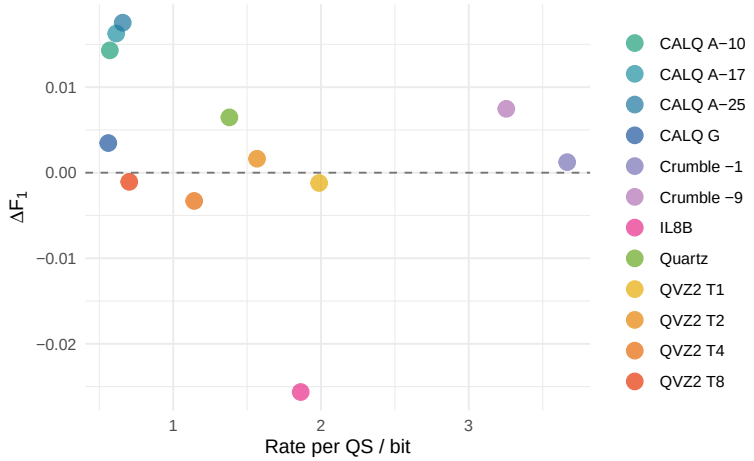


Figure A.9: F_1 score difference versus rate per QS, item 11, GATK VQSR 99.9%.

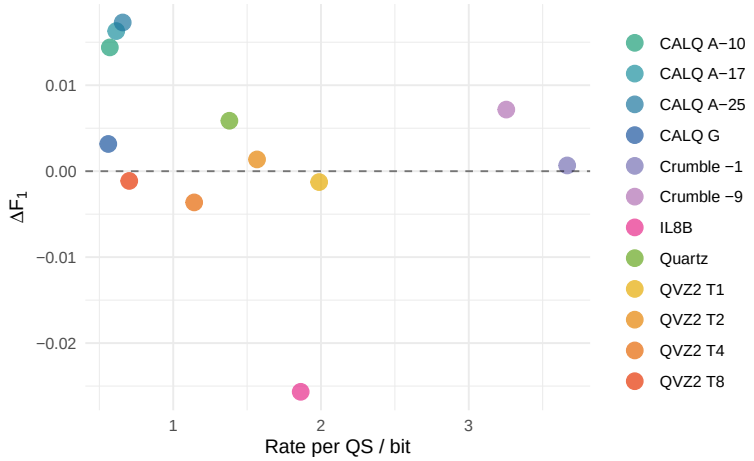


Figure A.10: F_1 score difference versus rate per QS, item 11, GATK VQSR 100.0%.

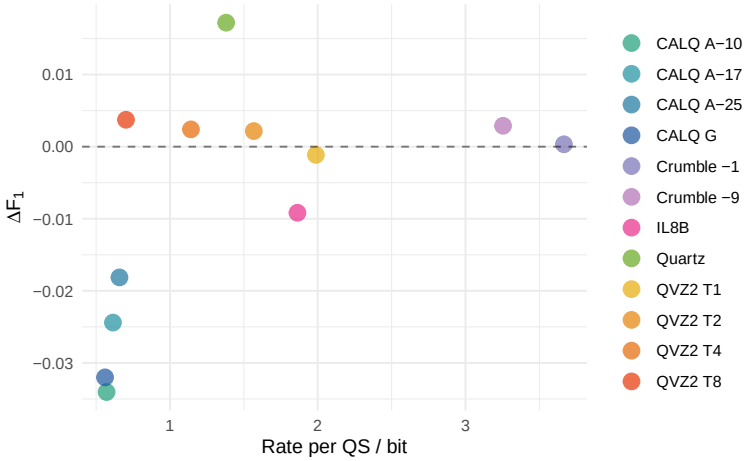


Figure A.11: F_1 score difference versus rate per QS, item 11, GATK HF.

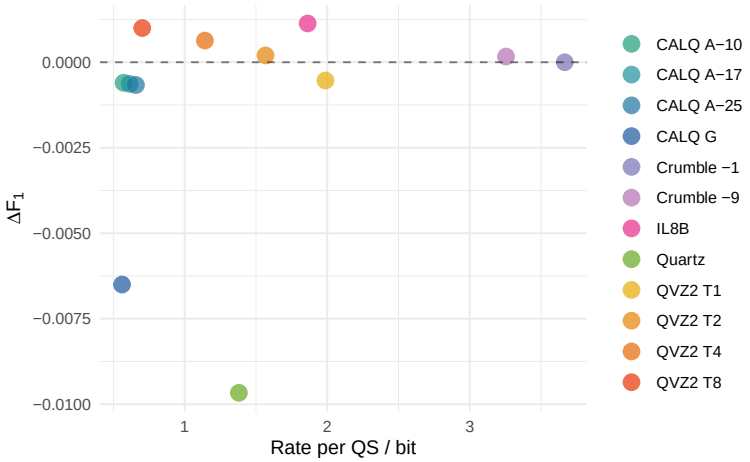


Figure A.12: F_1 score difference versus rate per QS, item 11, Platypus.

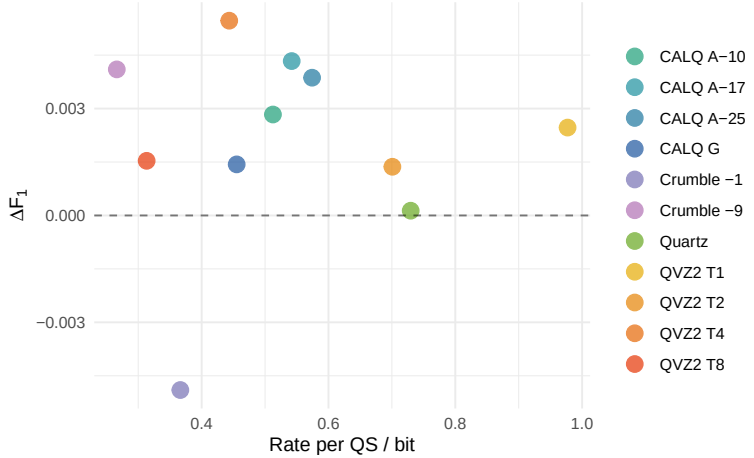


Figure A.13: F_1 score difference versus rate per QS, item 12, GATK VQSR 90.0%.

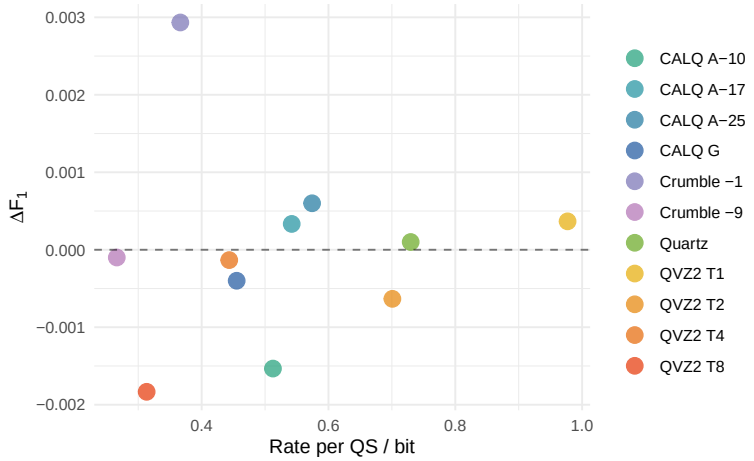


Figure A.14: F_1 score difference versus rate per QS, item 12, GATK VQSR 99.0%.

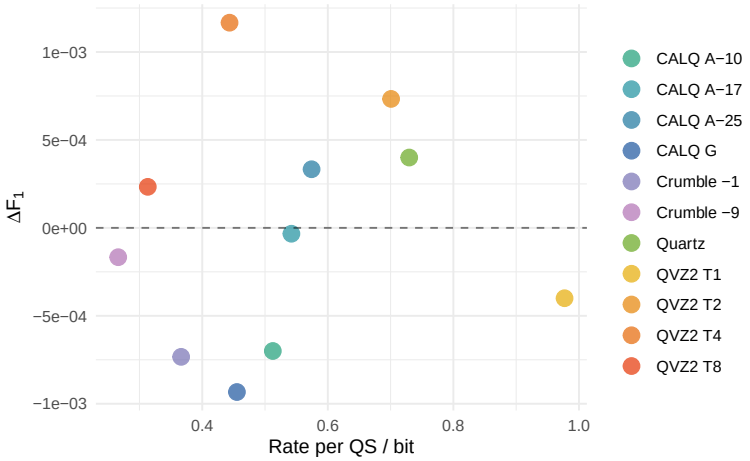


Figure A.15: F_1 score difference versus rate per QS, item 12, GATK VQSR 99.9%.

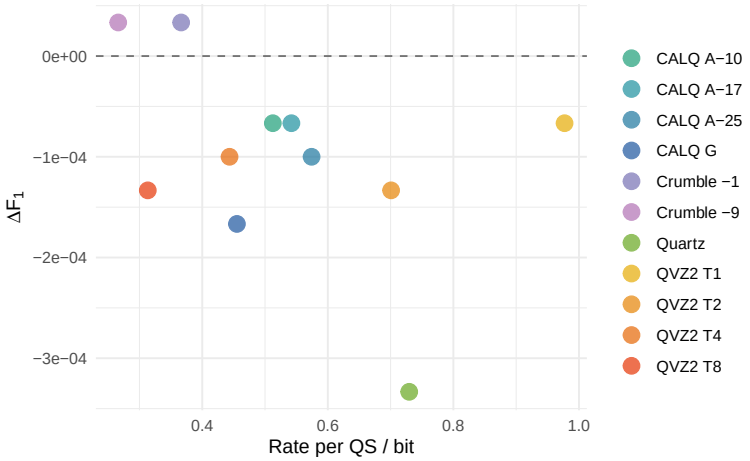


Figure A.16: F_1 score difference versus rate per QS, item 12, GATK VQSR 100.0%.

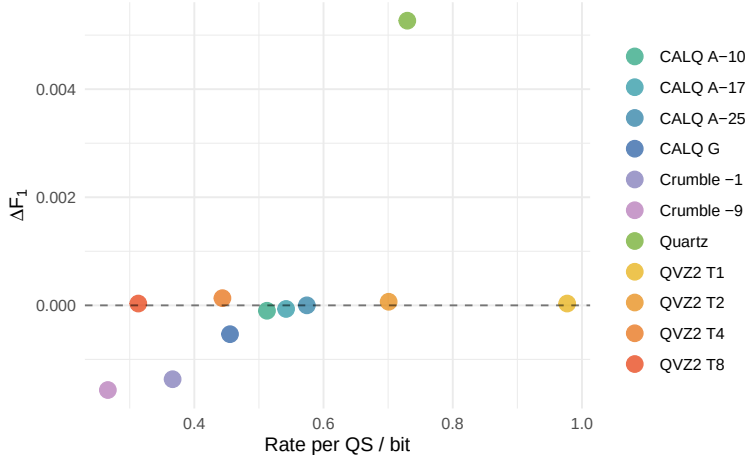


Figure A.17: F_1 score difference versus rate per QS, item 12, GATK HF.

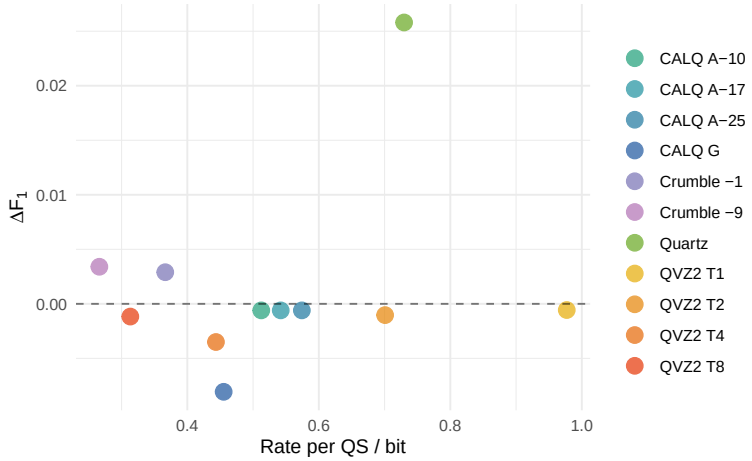


Figure A.18: F_1 score difference versus rate per QS, item 12, Platypus .

BIBLIOGRAPHY

[Ben+15] G. Benoit, C. Lemaitre, D. Lavenier, E. Drezen, T. Dayris, R. Uricaru, and G. Rizk. "Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph." In: *BMC Bioinformatics* 16.1 (2015), p. 288. DOI: 10.1186/s12859-015-0709-7.

[Bla+97] F. R. Blattner et al. "The Complete Genome Sequence of Escherichia coli K-12." In: *Science* 277.5331 (1997), pp. 1453–1462. DOI: 10.1126/science.277.5331.1453.

[Bon14] J. K. Bonfield. "The Scramble conversion tool." In: *Bioinformatics* 30.19 (2014), pp. 2818–2819. DOI: 10.1093/bioinformatics/btu390.

[BM13] J. K. Bonfield and M. V. Mahoney. "Compression of FASTQ and SAM Format Sequencing Data." In: *PLOS ONE* 8.3 (2013), e59190. DOI: 10.1371/journal.pone.0059190.

[BMD18] J. K. Bonfield, S. A. McCarthy, and R. Durbin. "Crumble: reference free lossy compression of sequence quality values." In: *Bioinformatics* 35.2 (2018), pp. 337–339. DOI: 10.1093/bioinformatics/bty608.

[BW10] J. K. Bonfield and A. Whitwham. "Gap5—editing the billion fragment sequence assembly." In: *Bioinformatics* 26.14 (2010), pp. 1699–1703. DOI: 10.1093/bioinformatics/btq268.

[CMT14] R. Cánovas, A. Moffat, and A. Turpin. "Lossy compression of quality scores in genomic data." In: *Bioinformatics* 30.15 (2014), pp. 2130–2136. DOI: 10.1093/bioinformatics/btu183.

[Cha+19] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez, and T. Weissman. "SPRING: a next-generation compressor for FASTQ data." In: *Bioinformatics* 35.15 (2019), pp. 2674–2676. DOI: 10.1093/bioinformatics/bty1015.

[CTW18] S. Chandak, K. Tatwawadi, and T. Weissman. "Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis." In: *Bioinformatics* 34.4 (2018), pp. 558–567. DOI: 10.1093/bioinformatics/btx639.

[Che+02] X. Chen, M. Li, B. Ma, and J. Tromp. "DNACompress: fast and effective DNA sequence compression." In: *Bioinformatics* 18.12 (2002), pp. 1696–1698. DOI: 10.1093/bioinformatics/18.12.1696.

[Chi+13] C.-S. Chin et al. "Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data." In: *Nature Methods* 10.6 (2013), pp. 563–569. DOI: 10.1038/nmeth.2474.

[Chr+09] S. Christley, Y. Lu, C. Li, and X. Xie. "Human genomes as email attachments." In: *Bioinformatics* 25.2 (2009), pp. 274–275. DOI: 10.1093/bioinformatics/btn582.

[Coc+10] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants." In: *Nucleic Acids Research* 38.6 (2010), pp. 1767–1771. DOI: 10.1093/nar/gkp1137.

[Cox+12] A. J. Cox, M. J. Bauer, T. Jakobi, and G. Rosone. "Large-scale compression of genomic sequence databases with the Burrows–Wheeler transform." In: *Bioinformatics* 28.11 (2012), pp. 1415–1419. DOI: 10.1093/bioinformatics/bts173.

[Cri58] F. H. C. Crick. "On protein synthesis." In: *Symposia of the Society for Experimental Biology* 12 (1958), pp. 138–163.

[Cri70] F. H. C. Crick. "Central Dogma of Molecular Biology." In: *Nature* 227 (1970), pp. 561–563. DOI: 10.1038/227561a0.

[Cri88] F. H. C. Crick. *What Mad Pursuit: A Personal View of Scientific Discovery*. Basic Books, 1988.

[DDN15] S. Deorowicz, A. Danek, and M. Niemiec. "GDC 2: Compression of large collections of genomes." In: *Scientific Reports* 5.1 (2015), p. 11565. DOI: 10.1038/srep11565.

[DG11a] S. Deorowicz and S. Grabowski. "Compression of DNA sequence reads in FASTQ format." In: *Bioinformatics* 27.6 (2011), pp. 860–862. DOI: 10.1093/bioinformatics/btr014.

[DG11b] S. Deorowicz and S. Grabowski. "Robust relative compression of genomes with random access." In: *Bioinformatics* 27.21 (2011), pp. 2979–2986. DOI: 10.1093/bioinformatics/btr505.

[Deo+15] S. Deorowicz, S. Grabowski, I. Ochoa, M. Hernaez, and T. Weissman. "Comment on: 'ERGC: an efficient referential genome compression algorithm'." In: *Bioinformatics* 32.7 (2015), pp. 1115–1117. DOI: 10.1093/bioinformatics/btv704.

[DeP+11] M. A. DePristo et al. "A framework for variation discovery and genotyping using next-generation DNA sequencing data." In: *Nature Genetics* 43.5 (2011), pp. 491–498. DOI: 10.1038/ng.806.

[Deu96] L. P. Deutsch. *GZIP file format specification version 4.3*. 1996. URL: https://tools.ietf.org/html/rfc1952 (visited on 02/22/2021).

[Duc+07] M. Duc Cao, T. I. Dix, L. Allison, and C. Mears. "A Simple Statistical Algorithm for Biological Sequence Compression." In: *2007 Data Compression Conference (DCC)*. 2007, pp. 43–52. DOI: 10.1109/DCC.2007.7.

[Dud14] J. Duda. *Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding*. 2014. arXiv: 1311.2540v2.

[Dut+15] A. Dutta, M. M. Haque, T. Bose, C. V. S. K. Reddy, and S. S. Mande. "FQC: A novel approach for efficient compression, archival, and dissemination of fastq datasets." In: *Journal of Bioinformatics and Computational Biology* 13.03 (2015), p. 1541003. DOI: 10.1142/S0219720015410036.

[Eli75] P. Elias. "Universal codeword sets and representations of the integers." In: *IEEE Transactions on Information Theory* 21.2 (1975), pp. 194–203. DOI: 10.1109/TIT.1975.1055349.

[EG98] B. Ewing and P. Green. "Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities." In: *Genome Research* 8.3 (1998), pp. 186–194. DOI: 10.1101/gr.8.3.186.

[Fri+11] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney. "Efficient storage of high throughput DNA sequencing data using reference-based compression." In: *Genome Research* 21.5 (2011), pp. 734–740. DOI: 10.1101/gr.114819.110.

[GV75] R. Gallager and D. van Voorhis. "Optimal source codes for geometrically distributed integer alphabets (Corresp.)" In: *IEEE Transactions on Information Theory* 21.2 (1975), pp. 228–230. DOI: 10.1109/TIT.1975.1055357.

[Gol66] S. Golomb. "Run-length encodings (Corresp.)" In: *IEEE Transactions on Information Theory* 12.3 (1966), pp. 399–401. DOI: 10.1109/TIT.1966.1053907.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. The MIT Press, 2016.

[GMM16] S. Goodwin, J. D. McPherson, and W. R. McCombie. "Coming of age: ten years of next-generation sequencing technologies." In: *Nature Reviews Genetics* 17.6 (2016), pp. 333–351. DOI: 10.1038/nrg.2016.49.

[Goy+20] M. Goyal, K. Tatwawadi, S. Chandak, and I. Ochoa. *DZip: improved general-purpose lossless compression based on novel neural network modeling*. 2020. arXiv: 1911.03572.

[GDR15] S. Grabowski, S. Deorowicz, and Ł. Roguski. "Disk-based compression of data from genome sequencing." In: *Bioinformatics* 31.9 (2015), pp. 1389–1395. DOI: 10.1093/bioinformatics/btu844.

[GSR16] D. L. Greenfield, O. Stegle, and A. Rrustemi. "GeneCodeq: quality score compression and improved genotyping using a Bayesian framework." In: *Bioinformatics* 32.20 (2016), pp. 3124–3132. DOI: 10.1093/bioinformatics/btw385.

[HNS14] F. Hach, I. Numanagic, and S. C. Sahinalp. "DeeZ: reference-based compression by local assembly." In: *Nature Methods* 11.11 (2014), pp. 1082–1084. DOI: 10.1038/nmeth.3133.

[Hac+12] F. Hach, I. Numanagić, C. Alkan, and S. C. Sahinalp. "SCALCE: boosting sequence compression algorithms using locally consistent encoding." In: *Bioinformatics* 28.23 (2012), pp. 3051–3057. DOI: 10.1093/bioinformatics/bts593.

[HOW16] M. Hernaez, I. Ochoa, and T. Weissman. "A Cluster-Based Approach to Compression of Quality Scores." In: *2016 Data Compression Conference (DCC)*. 2016, pp. 261–270. DOI: 10.1109/DCC.2016.49.

[Her+19] M. Hernaez, D. Pavlichin, T. Weissman, and I. Ochoa. "Genomic Data Compression." In: *Annual Review of Biomedical Data Science* 2 (2019), pp. 19–37. DOI: 10.1146/annurev-biodatasci-072018-021229.

[Huf52] D. A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes." In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101. DOI: 10.1109/JRPROC.1952.273898.

[Int01] International Human Genome Sequencing Consortium. "Initial sequencing and analysis of the human genome." In: *Nature* 409.6822 (2001), pp. 860–921. DOI: 10.1038/35057062.

[ISO15a] ISO/IEC JTC1/SC29/WG11. *Call for Evidence for Genome Compression and Storage*. N15740. 2015.

[ISO15b] ISO/IEC JTC1/SC29/WG11. *Requirements on Genome Compression and Storage*. N15738. 2015.

[ISO16a] ISO/IEC JTC1/SC29/WG11. *Evaluation Procedure for the Joint Call for Proposals on Genomic Information Compression and Storage*. N16321. 2016.

[ISO16b] ISO/IEC JTC1/SC29/WG11. *Joint Call for Proposals for Genomic Information Compression and Storage*. N16320. 2016.

[ISO16c] ISO/IEC JTC1/SC29/WG11. *Summary of the current status and workplan of the joint TC276/WG5 JTC1/SC29/WG11 standardization activities on genomic information representation*. N16529. 2016.

[ISO20] ISO/IEC JTC1/SC29/WG11. *MPEG-G Genomic Information Database*. N19311. 2020.

[Jay+74] E. Jay, R. Bambara, R. Padmanabhan, and R. Wu. "DNA sequence analysis: a general, simple and rapid method for sequencing large oligodeoxyribonucleotide fragments by mapping*." In: *Nucleic Acids Research* 1.3 (1974), pp. 331–354. DOI: 10.1093/nar/1.3.331.

[Jon+12] D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze. "Compression of next-generation sequencing reads aided by highly efficient de novo assembly." In: *Nucleic Acids Research* 40.22 (2012), e171. DOI: 10.1093/nar/gks754.

[Kel+19] J. Kelleher et al. "htsget: a protocol for securely streaming genomic data." In: *Bioinformatics* 35.1 (2019), pp. 119–121. DOI: 10.1093/bioinformatics/bty492.

[KP15] C. Kingsford and R. Patro. "Reference-based compression of short-read sequences using path encoding." In: *Bioinformatics* 31.12 (2015), pp. 1920–1928. DOI: 10.1093/bioinformatics/btv071.

[Kol33] A. N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer, 1933.

[Kru+19] P. Krusche et al. "Best practices for benchmarking germline small-variant calls in human genomes." In: *Nature Biotechnology* 37.5 (2019), pp. 555–560. DOI: 10.1038/s41587-019-0054-x.

[Lei+11] R. Leinonen, H. Sugawara, M. Shumway, and on behalf of the International Nucleotide Sequence Database Collaboration. "The Sequence Read Archive." In: *Nucleic Acids Research* 39 (suppl_1 2011), pp. D19–D21. DOI: 10.1093/nar/gkq1019.

[LD09] H. Li and R. Durbin. "Fast and accurate short read alignment with Burrows–Wheeler transform." In: *Bioinformatics* 25.14 (2009), pp. 1754–1760. DOI: 10.1093/bioinformatics/btp324.

[Li+09] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup. "The Sequence Alignment/Map format and SAMtools." In: *Bioinformatics* 25.16 (2009), pp. 2078–2079. DOI: 10.1093/bioinformatics/btp352.

[Liu+12] L. Liu, Y. Li, S. Li, N. Hu, Y. He, R. Pong, D. Lin, L. Lu, and M. Law. "Comparison of Next-Generation Sequencing Systems." In: *BioMed Research International* 2012 (2012), p. 251364. DOI: 10.1155/2012/251364.

[Lod+07] H. F. Lodish, A. Berk, C. A. Kaiser, M. Krieger, and M. P. Scott. *Molecular Cell Biology*. W. H. Freeman and Company, 2007.

[Mal+15] G. Malysa, M. Hernaez, I. Ochoa, M. Rao, K. Ganesan, and T. Weissman. "QVZ: lossy compression of quality values." In: *Bioinformatics* 31.19 (2015), pp. 3122–3129. DOI: 10.1093/bioinformatics/btv330.

[MSW03] D. Marpe, H. Schwarz, and T. Wiegand. "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard." In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (2003), pp. 620–636. DOI: 10.1109/TCSVT.2003.815173.

[MG77] A. M. Maxam and W. Gilbert. "A new method for sequencing DNA." In: *Proceedings of the National Academy of Sciences* 74.2 (1977), pp. 560–564. DOI: 10.1073/pnas.74.2.560.

[McK+10] A. McKenna et al. "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data." In: *Genome Research* 20.9 (2010), pp. 1297–1303. DOI: 10.1101/gr.107524.110.

[Muso2] H.-G. Musmann. *Quellencodierung*. Lecture Notes. Institut für Informationsverarbeitung, Leibniz Universität Hannover, 2002.

[Nar20] D. Naro. "Security strategies in genomic files." PhD thesis. Universitat Politècnica de Catalunya, 2020.

[NPR15] M. Nicolae, S. Pathak, and S. Rajasekaran. "LFQC: a lossless compression algorithm for FASTQ files." In: *Bioinformatics* 31.20 (2015), pp. 3276–3281. DOI: 10.1093/bioinformatics/btv384.

[Nob21a] Nobel Prize Outreach AB. *The Nobel Prize in Chemistry 1958*. 2021. URL: https://NobelPrize.org (visited on 09/14/2021).

[Nob21b] Nobel Prize Outreach AB. *The Nobel Prize in Chemistry 1980*. 2021. URL: https://NobelPrize.org (visited on 09/14/2021).

[Och+13] I. Ochoa, H. Asnani, D. Bharadia, M. Chowdhury, T. Weissman, and G. Yona. "QualComp: a new lossy compressor for quality scores based on rate distortion theory." In: *BMC Bioinformatics* 14.1 (2013), p. 187. DOI: 10.1186/1471-2105-14-187.

[Och+17] I. Ochoa, M. Hernaez, R. Goldfeder, T. Weissman, and E. Ashley. "Effect of lossy compression of quality scores on variant calling." In: *Briefings in Bioinformatics* 18.2 (2017), pp. 183–194. DOI: 10.1093/bib/bbw011.

[OHW15] I. Ochoa, M. Hernaez, and T. Weissman. "iDoComp: a compression scheme for assembled genomes." In: *Bioinformatics* 31.5 (2015), pp. 626–633. DOI: 10.1093/bioinformatics/btu698.

[PW72] R. Padmanabhan and R. Wu. "Nucleotide sequence analysis of DNA: IX. Use of oligonucleotides of defined sequence as primers in DNA sequence analysis." In: *Biochemical and Biophysical Research Communications* 48.5 (1972), pp. 1295–1302. DOI: 10.1016/0006-291X(72)90852-2.

[Par+17] T. Paridaens, G. Van Wallendael, W. De Neve, and P. Lambert. "AFRESh: an adaptive framework for compression of reads and assembled sequences with random access functionality." In: *Bioinformatics* 33.10 (2017), pp. 1464–1472. DOI: 10.1093/bioinformatics/btx001.

[Par+18] T. Paridaens, G. Van Wallendael, W. De Neve, and P. Lambert. "AQUa: an adaptive framework for compression of sequencing quality scores with random access functionality." In: *Bioinformatics* 34.3 (2018), pp. 425–433. DOI: 10.1093/bioinformatics/btx607.

[PK15] R. Patro and C. Kingsford. "Data-dependent bucketing improves reference-free compression of sequencing reads." In: *Bioinformatics* 31.17 (2015), pp. 2770–2777. DOI: 10.1093/bioinformatics/btv248.

[Pay+19] A. Payne, N. Holmes, V. Rakyan, and M. Loose. "BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files." In: *Bioinformatics* 35.13 (2019), pp. 2193–2198. DOI: 10.1093/bioinformatics/bty841.

[PPG11] A. J. Pinho, D. Pratas, and S. P. Garcia. "GReEn: a tool for efficient compression of genome resequencing data." In: *Nucleic Acids Research* 40.4 (2011), e27. DOI: 10.1093/nar/gkr1124.

[Pop+18] R. Poplin et al. "Scaling accurate genetic variant discovery to tens of thousands of samples." In: *bioRxiv* 201178 (2018). [not peer-reviewed]. DOI: 10.1101/201178.

[Qua+12] M. A. Quail, M. Smith, P. Coupland, T. D. Otto, S. R. Harris, T. R. Connor, A. Bertoni, H. P. Swerdlow, and Y. Gu. "A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers." In: *BMC Genomics* 13 (2012), p. 341. DOI: 10.1186/1471-2164-13-341.

[RTP73] W. Ray, C.-p. D. Tu, and R. Padmanabhan. "Nucleotide sequence analysis of DNA XII. The chemical synthesis and sequence analysis of a dodecadeoxynucleotide which binds to the endolysin gene of bacteriophage lambda." In: *Biochemical and Biophysical Research Communications* 55.4 (1973), pp. 1092–1099. DOI: 10.1016/S0006-291X(73)80007-5.

[RP71] R. Rice and J. Plaunt. "Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data." In: *IEEE Transactions on Communication Technology* 19.6 (1971), pp. 889–897. DOI: 10.1109/TCOM.1971.1090789.

[Rim+14] A. Rimmer, H. Phan, I. Mathieson, Z. Iqbal, S. R. F. Twigg, A. O. M. Wilkie, G. McVean, G. Lunter, and WGS500 Consortium. "Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications." In: *Nature Genetics* 46.8 (2014), pp. 912–918. DOI: 10.1038/ng.3036.

[RD14] Ł. Roguski and S. Deorowicz. "DSRC 2—Industry-oriented compression of FASTQ files." In: *Bioinformatics* 30.15 (2014), pp. 2213–2215. DOI: 10.1093/bioinformatics/btu208.

[Rog+18] Ł. Roguski, I. Ochoa, M. Hernaez, and S. Deorowicz. "FaStore: a space-saving solution for raw sequencing data." In: *Bioinformatics* 34.16 (2018), pp. 2748–2756. DOI: 10.1093/bioinformatics/bty205.

[SR15] S. Saha and S. Rajasekaran. "ERGC: an efficient referential genome compression algorithm." In: *Bioinformatics* 31.21 (2015), pp. 3468–3475. DOI: 10.1093/bioinformatics/btv399.

[San+77] F. Sanger, G. M. Air, B. G. Barrell, N. L. Brown, A. R. Coulson, J. C. Fiddes, C. A. Hutchison, P. M. Slocombe, and M. Smith. "Nucleotide sequence of bacteriophage ΦX174 DNA." In: *Nature* 265.5596 (1977), pp. 687–695. DOI: 10.1038/265687a0.

[San+82] F. Sanger, A. R. Coulson, G. F. Hong, D. F. Hill, and G. B. Petersen. "Nucleotide sequence of bacteriophage λ DNA." In: *Journal of Molecular Biology* 162.4 (1982), pp. 729–773. DOI: 10.1016/0022-2836(82)90546-0.

[SNC77] F. Sanger, S. Nicklen, and A. R. Coulson. "DNA sequencing with chain-terminating inhibitors." In: *Proceedings of the National Academy of Sciences* 74.12 (1977), pp. 5463–5467. DOI: 10.1073/pnas.74.12.5463.

[Say06] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2006.

[Ste+15] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson. "Big Data: Astronomical or Genomical?" In: *PLOS Biology* 13.7 (2015), e1002195. DOI: 10.1371/journal.pbio.1002195.

[Sul+12] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. "Overview of the High Efficiency Video Coding (HEVC) Standard." In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (2012), pp. 1649–1668. DOI: 10.1109/TCSVT.2012.2221191.

[The10] The 1000 Genomes Project Consortium. "A map of human genome variation from population-scale sequencing." In: *Nature* 467.7319 (2010), pp. 1061–1073. DOI: 10.1038/nature09534.

[The21] The SAM/BAM Format Specification Working Group. *Sequence Alignment/Map Format Specification*. 2021. URL: https://samtools.github.io/hts-specs/SAMv1.pdf (visited on 02/22/2021).

[VO20] G. A. Van der Auwera and B. D. O'Connor. *Genomics in the Cloud: Using Docker, GATK, and WDL in Terra*. O'Reilly Media, 2020.

[WZ11] C. Wang and D. Zhang. "A novel compression tool for efficient storage of genome resequencing data." In: *Nucleic Acids Research* 39.7 (2011), e45. DOI: 10.1093/nar/gkr009.

[WC53] J. D. Watson and F. H. C. Crick. "Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid." In: *Nature* 171.4356 (1953), pp. 737–738. DOI: 10.1038/171737a0.

[Wie+03] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra. "Overview of the H.264/AVC video coding standard." In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (2003), pp. 560–576. DOI: 10.1109/TCSVT.2003.815165.

[WNC87] I. H. Witten, R. M. Neal, and J. G. Cleary. "Arithmetic coding for data compression." In: *Communications of the ACM* 30.6 (1987), pp. 520–540. DOI: 10.1145/214762.214771.

[Yat+21] A. D. Yates, J. Adams, S. Chaturvedi, R. M. Davies, M. Laird, R. Leinonen, R. Nag, N. C. Sheffield, O. Hofmann, and T. Keane. "Refget: standardised access to reference sequences." In: *bioRxiv* 2021.03.11.434800 (2021). [not peer-reviewed]. DOI: 10.1101/2021.03.11.434800.

[Yu+15] Y. W. Yu, D. Yorukoglu, J. Peng, and B. Berger. "Quality score compression improves genotyping accuracy." In: *Nature Biotechnology* 33.3 (2015), pp. 240–243. DOI: 10.1038/nbt.3170.

[Zha+16] Y. Zhang, K. Patel, T. Endrawis, A. Bowers, and Y. Sun. "A FASTQ compressor based on integer-mapped k-mer indexing for biologist." In: *Gene* 579.1 (2016), pp. 75–81. DOI: 10.1016/j.gene.2015.12.053.

[Zha+15] Y. Zhang, L. Li, Y. Yang, X. Yang, S. He, and Z. Zhu. "Lightweight reference-based compression of FASTQ data." In: *BMC Bioinformatics* 16.1 (2015), p. 188. DOI: 10.1186/s12859-015-0628-7.

[ZL77] J. Ziv and A. Lempel. "A universal algorithm for sequential data compression." In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343. DOI: 10.1109/TIT.1977.1055714.

[Zoo+14] J. M. Zook, B. Chapman, J. Wang, D. Mittelman, O. Hofmann, W. Hide, and M. Salit. "Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls." In: *Nature Biotechnology* 32.3 (2014), pp. 246–251. DOI: 10.1038/nbt.2835.

PUBLICATIONS

JOURNAL ARTICLES

[Lau+19] T. Laude, Y. G. Adhisantoso, J. Voges, M. Munderloh, and J. Ostermann. "A Comprehensive Video Codec Comparison." In: *APSIPA Transactions on Signal and Information Processing* 8 (2019), e30. DOI: 10.1017/ATSIP.2019.23.

[Num+16] I. Numanagić, J. K. Bonfield, F. Hach, J. Voges, J. Ostermann, C. Alberti, M. Mattavelli, and S. C. Sahinalp. "Comparison of high-throughput sequencing data compression tools." In: *Nature Methods* 13.12 (2016), pp. 1005–1008. DOI: 10.1038/nmeth.4037.

[Vog+18b] J. Voges, A. Fotouhi, J. Ostermann, and M. O. Külekci. "A Two-Level Scheme for Quality Score Compression." In: *Journal of Computational Biology* 25.10 (2018), pp. 1141–1151. DOI: 10.1089/cmb.2018.0065.

[Vog+21] J. Voges, M. Hernaez, M. Mattavelli, and J. Ostermann. "An Introduction to MPEG-G: The First Open ISO/IEC Standard for the Compression and Exchange of Genomic Sequencing Data." In: *Proceedings of the IEEE* 109.9 (2021), pp. 1607–1622. DOI: 10.1109/JPROC.2021.3082027.

[VOH18] J. Voges, J. Ostermann, and M. Hernaez. "CALQ: compression of quality values of aligned sequencing data." In: *Bioinformatics* 34.10 (2018), pp. 1650–1658. DOI: 10.1093/bioinformatics/btx737.

[Vog+20] J. Voges, T. Paridaens, F. Müntefering, L. S. Mainzer, B. Bliss, M. Yang, I. Ochoa, J. Fostier, J. Ostermann, and M. Hernaez. "GABAC: an arithmetic coding solution for genomic data." In: *Bioinformatics* 36.7 (2020), pp. 2275–2277. DOI: 10.1093/bioinformatics/btz922.

CONFERENCE CONTRIBUTIONS

Papers

[Alb+16] C. Alberti, N. Daniels, M. Hernaez, J. Voges, R. L. Goldfeder, A. A. Hernandez-Lopez, M. Mattavelli, and B. Berger. "An Evaluation Framework for Lossy Compression of Genome Sequencing Quality Values." In: *2016 Data Compression Conference (DCC)*. 2016, pp. 221–230. DOI: 10.1109/DCC.2016.39.

[Her+18] A. A. Hernandez-Lopez, J. Voges, C. Alberti, M. Mattavelli, and J. Ostermann. "Lossy Compression of Quality Scores in Differential Gene Expression: A First Assessment and Impact Analysis." In: *2018 Data Compression Conference (DCC)*. 2018, pp. 167–176. DOI: 10.1109/DCC.2018.00025.

[Lau+18] T. Laude, Y. G. Adhisantoso, J. Voges, M. Munderloh, and J. Ostermann. "A Comparison of JEM and AV1 with HEVC: Coding Tools, Coding Efficiency and Complexity." In: *2018 Picture Coding Symposium (PCS)*. 2018, pp. 36–40. DOI: 10.1109/PCS.2018.8456291.

[Och+19] I. Ochoa, H. Li, F. Baumgarte, C. Hergenrother, J. Voges, and M. Hernaez. "AliCo: A New Efficient Representation for SAM Files." In: *2019 Data Compression Conference (DCC)*. 2019, pp. 93–102. DOI: 10.1109/DCC.2019.00017.

[Vog+18a] J. Voges, A. Fotouhi, J. Ostermann, and M. O. Külekci. "A Two-Level Scheme for Quality Score Compression." In: *10th International Conference on Bioinformatics and Computational Biology (BICOB 2018)*. 2018, pp. 93–102. URL: http://www.proceedings.com/38658.html.

[VMO16] J. Voges, M. Munderloh, and J. Ostermann. "Predictive Coding of Aligned Next-Generation Sequencing Data." In: *2016 Data Compression Conference (DCC)*. 2016, pp. 241–250. DOI: 10.1109/DCC.2016.98.

Posters

[Her+17] A. A. Hernandez-Lopez, J. Voges, C. Alberti, M. Mattavelli, and J. Ostermann. "Differential Gene Expression with Lossy Compression of Quality Scores in RNA-Seq Data." In: *2017 Data Compression Conference (DCC)*. 2017, p. 444. DOI: 10.1109/DCC.2017.75.

[Par+19] T. Paridaens, J. Voges, M. Hernaez, J. Fostier, and J. Ostermann. "GABAC: an arithmetic coding solution for genomic data." In: *F1000Research* 8 (International Society for Computational Biology Community Journal) (2019). [version 1; not peer-reviewed], p. 1463. DOI: 10.7490/f1000research.1117346.1.

[VO17] J. Voges and J. Ostermann. "MPEG-G: The Emerging Standard for Genomic Data." In: *Poster abstracts of the 25th German Conference on Bioinformatics*. [not peer-reviewed]. 2017, p. 2. DOI: 10.7287/peerj.preprints.3268v1.

[VOH17] J. Voges, J. Ostermann, and M. Hernaez. "CALQ: compression of quality values of aligned sequencing data." In: *F1000Research* 6 (International Society for Computational Biology Community Journal) (2017). [version 1; not peer-reviewed], p. 1382. DOI: 10.7490/f1000research.1114635.1.

PREPRINTS

[Alb+18] C. Alberti et al. "An introduction to MPEG-G, the new ISO standard for genomic information representation." In: *bioRxiv* 426353 (2018). [not peer-reviewed]. DOI: 10.1101/426353.

TECHNICAL REPORTS

[OV19] J. Ostermann and J. Voges. "Streaming für die Genomforschung." In: *Binaire* 2019.2 (2019). [not peer-reviewed]. URL: https://www.l3s.de/de/projects/coding-sequencing-data.

PATENTS

[VOa] J. Voges and J. Ostermann. "Method for encoding and decoding of quality values of a data structure." EP3652862B1, US10938415B2.

PATENT APPLICATIONS

[MVO] M. Munderloh, J. Voges, and J. Ostermann. "Method for compressing genomic data." CN107851137A, EP3311318A1, US20180181706A1.

[VHO] J. Voges, M. Hernaez, and J. Ostermann. "Method for encoding and decoding of quality values of a data structure." CN110168650A, EP3526708A1, US20210295950A1.

[VOb] J. Voges and J. Ostermann. "Method for encoding and decoding of quality values of a data structure." CN110915140A.

[Vog+] J. Voges, C. Rohlfing, V. Tunev, Y. G. Adhisantoso, and J. Ostermann. "Verfahren zum Komprimieren, Verfahren zum Dekomprimieren einer Information einer Erbinformation, elektronische Speichereinrichtung sowie Datenverarbeitungssystem." DE102021100199A1.

STANDARDIZATION CONTRIBUTIONS

ISO/IEC JTC1/SC29/WG11

m36282—Jan Voges and Marco Munderloh. "Approaches To SAM File Compression."

m37061—Jan Voges. "A Framework for the Evaluation of Genomic Information Compression."

m37678—Jan Voges, Marco Munderloh, and Jörn Ostermann. "Answer to the Call for Evidence for Genome Compression and Storage: Reference-free Compression of Aligned Next-Generation Sequencing Data."

m38376—Jan Voges, Mikel Hernaez, Ana A. Hernandez-Lopez, Claudio Alberti, Marco Mattavelli, Al Wegener, Dan Greenfield, Noah Daniels, S. Cenk Sahinalp, James Bonfield, and Bonnie Berger. "Extensions and notes to the evaluation framework for lossy compression of genome sequencing quality values."

m38916—Jan Voges, Mikel Hernaez, Idoia Ochoa, Ana A. Hernandez-Lopez, Claudio Alberti, Marco Mattavelli, Al Wegener, Dan Greenfield, Noah Daniels, S. Cenk Sahinalp, James Bonfield, Bonnie Berger, and Jörn Ostermann. "Benchmark framework for lossy compression of genome sequencing quality values."

m38917—Jan Voges, Mikel Hernaez, and Jörn Ostermann. "Adaptive lossy compression of high-throughput sequencing quality values."

m38918—Jan Voges, Marco Munderloh, Jörn Ostermann. "Reference-free compression of aligned high-throughput sequencing data."

m39663—Jan Voges. "Core Experiment 2 summary."

m39664—Jan Voges and Mikel Hernaez. "Core Experiment 2 results LUH-Stanford/UIUC."

m39665—Jan Voges. "Core Experiment 1 results LUH."

m39666—Jan Voges. "Core Experiment 3 results LUH."

m39748—Jan Voges. "Core Experiment 1 cross-check SFU."

m39749—Jan Voges. "Core Experiment 2 cross-check SFU."

m39750—Jan Voges. "Core Experiment 3 cross-check SFU."

m40222—Jan Voges, Claudio Alberti, Mikel Hernaez, Tom Paridaens, James Bonfield, Paolo Ribeca, and Jaime Delgado. "Unified representation of sequencing quality values."

m40223—Jan Voges. "Summary of Core Experiment 2 on Genomic Information Representation."

m40224—Jan Voges and Mikel Hernaez. "Core Experiment 2 on Genomic Information Representation results LUH/Stanford/UIUC."

m40225—Jan Voges. "Core Experiment 5 on Genomic Information Representation results LUH."

m40804—Jan Voges. "Core Experiment 5 on Genomic Information Representation results LUH."

m40860—Jan Voges. "Proposed changes for ISO/IEC 23092-2 WD 2."

m40861—Jan Voges. "A Rate-Distortion Analysis of Sequencing Quality Value Compression."

m41605—Jan Voges. "Core Experiment 5 on Genomic Representation results LUH."

m41956—Jan Voges. "Study on White paper on the objectives and benefits of the MPEG-G standard (Draft 1)."

m42305—Jan Voges. "Study of ISO/IEC 23092-2."

m42306—Jan Voges. "Quality value coding and functional equivalence of genomic analysis pipelines."

m42919—Jan Voges. "Study of ISO/IEC CD 23092-3."

m42920—Jan Voges, Idoia Ochoa and Mikel Hernaez. "Update to the genomic information database."

m44035—Jan Voges, Idoia Ochoa and Mikel Hernaez. "Proposed Updates to the MPEG-G Genomic Information Database."

m44049—Jan Voges, Shubham Chandak, and Mikel Hernaez. "Study on ISO/IEC DIS 23092-2."

m45266—Jan Voges, Idoia Ochoa, Mikel Hernaez, Fabian Müntefering, and Giorgio Zoia. "Proposed Updates to the MPEG-G Genomic Information Database."

m45267—Paolo Ribeca, Jan Voges, Tom Paridaens, Mikel Hernaez, Idoia Ochoa, Claudio Alberti, Marco Mattavelli, Giuseppe Codispoti, Jaime Delgado, and Daniel Naro. "Thoughts on future standardization activities in the area of genomic information representation."

m45294—Jan Voges, Tom Paridaens, Daniel Naro, Dmitry Repchevski, Jaime Delgado, and Mikel Hernaez. "Study on ISO/IEC 23092-2."

m45601—Daniel Naro, Jan Voges, Tom Paridaens, Dmitry Repchevski, Jaime Delgado, Paolo Ribeca, Idoia Ochoa, and Mikel Hernaez. "Comments on ISO/IEC DIS 23092-1 and ISO/IEC DIS 23092-2."

m46660—Jan Voges, Idoia Ochoa, Mikel Hernaez, and James Bonfield. "Proposed Updates to the MPEG-G Genomic Information Database."

m48100—Jan Voges, Idoia Ochoa, Mikel Hernaez, Giorgio Zoia, and Marco Mattavelli. "Proposed Updates to the MPEG-G Genomic Information Database."

m51192—Christian Rohlfing, Viktor Tunev, and Jan Voges. "Proposed Updates to the MPEG-G Genomic Information Database."

m52489—Jan Voges, Christian Rohlfing, Viktor Tunev, Yeremia G. Adhisantoso, and Jörn Ostermann. "Method for the coding of genomic variants."

m52977—Jan Voges and Giorgio Zoia. "Proposed Updates to the MPEG-G Genomic Information Database."

m52978—Jan Voges, Fabian Müntefering, Yeremia G. Adhisantoso, Junaid Ahmad, Shubham Chandak, Liudmila S. Mainzer, Mikel Hernaez, Idoia Ochoa, and Jörn Ostermann. "Draft Description of the MPEG-G Reference Encoder Software."

m53496—Yeremia G. Adhisantoso, Viktor Tunev, Jan Voges, and Christian Rohlfing. "MPEG-G Part 6 CE3 Results Leibniz University Hannover, RWTH Aachen University."

m53789—Jan Voges. "MPEG-G Part 6 CE3 Cross-Check of UIUC/University of Navarra Results."

ISO/IEC JTC1/SC29/WG8

m54889—Jan Voges, Fabian Müntefering, Yeremia G. Adhisantoso, Junaid Ahmad, Shubham Chandak, Liudmila S. Mainzer, Mikel Hernaez, Idoia Ochoa, Christian Rohlfing, and Jörn Ostermann. "Draft Text Reference Encoder Software."

m55355—Yeremia G. Adhisantoso, Jan Voges, Christian Rohlfing, and Jörn Ostermann. "Extension to Method for Coding of Genomic Variants."

m55356—Yeremia G. Adhisantoso, Jan Voges, Christian Rohlfing, and Jörn Ostermann. "Method for the Coding of Genotype Likelihood of Variants."

m55694—Jan Voges, Fabian Müntefering, Yeremia G. Adhisantoso, Junaid Ahmad, Shubham Chandak, Liudmila S. Mainzer, Mikel Hernaez, Idoia Ochoa, Christian Rohlfing, and Jörn Ostermann. "Draft Text Reference Encoder Software."

m56353—Jan Voges, Fabian Müntefering, Yeremia G. Adhisantoso, Junaid Ahmad, Shubham Chandak, Liudmila S. Mainzer, Mikel Hernaez, Idoia Ochoa, Christian Rohlfing, and Jörn Ostermann. "Draft Text Reference Encoder Software."

m57764—Jan Voges, Fabian Müntefering, Yeremia G. Adhisantoso, Junaid Ahmad, Shubham Chandak, Liudmila S. Mainzer, Mikel Hernaez, Idoia Ochoa, Christian Rohlfing, and Jörn Ostermann. "Draft Text Reference Encoder Software."

Jan Voges

Personal Data

Year of Birth:	1988
Place of Birth:	Hannover, Germany
Email:	voges@tnt.uni-hannover.de

Work Experience

Since 2022	Group Leader Institut für Informationsverarbeitung, Leibniz University Hannover
2015–2022	Research Assistant Institut für Informationsverarbeitung, Leibniz University Hannover
2018	Visiting Researcher Carl R. Woese Institute for Genomic Biology, University of Illinois at Urbana-Champaign, Urbana, IL (US)
2012–2013	Student Assistant Institut für Informationsverarbeitung, Leibniz University Hannover
2011	Student Intern Intel GmbH, Brunswick (DE)
2010–2011	Student Assistant Institute of Electrical Engineering and Measurement Technology, Leibniz University Hannover

Education

2009–2015	Study (Diploma Prog. in Electrical Engineering) Leibniz University Hannover *Degree: Diplom-Ingenieur*
2007–2009	Study (Bachelor of Science in Mechatronics) Leibniz University Hannover
2007	Abitur Matthias-Claudius-Gymnasium, Gehrden (DE)