Charles University
Faculty of Science

Study programme: Bioinformatics

Branch of study: Bioinformatics

Kateřina Břicháčková

Limitations of variant consequence predictors

Omezení predikčních programů pro určování důsledků genomických variant

Bachelor's thesis

Supervisor: Mgr. Petr Daněček, Ph.D.

Prague, 2018

Prohlášení

Prohlašuji, že jsem závěrečnou práci zpracovala samostatně a že jsem uvedla všechny použité informační zdroje a literaturu. Tato práce ani její podstatná část nebyla předložena k získání jiného nebo stejného akademického titulu.

V Praze, 6.8.2018 Kateřina Břicháčková

Poděkování

Na tomto místě bych ráda poděkovala svému školiteli Mgr. Petru Daněčkovi, Ph.D. za cenné rady, ochotu a trpělivost.

Mgr. Marianu Novotnému, Ph.D. velice děkuji, že ve mně dokázal vzbudit zájem o bioinformatiku a že vkládá tolik úsilí do toho, aby se náš (zatím) malý obor stále rozrůstal.

Největší dík patří mé rodině a příteli Davidu Jiříčkovi. Děkuji Vám, že tu pro mě vždy jste a že nikdy neztrácíte důvěru ve mě.

A nesmím zapomenout ani na svého psa Čendu, kterému děkuji za jeho věčný optimismus.

Acknowledgement

I would like to thank my supervisor, Mgr. Petr Daněček, Ph.D. for much helpful advice, kindliness and patient guidance throughout writing this thesis.

Many thanks to Mgr. Marian Novotný, Ph.D. for exciting my interest in bioinformatics and for putting so much effort in making the bioinformatics branch at our university grow.

The biggest thanks go to my family and partner David Jiříček. Thank you all that you are always here for me and you never lose faith in me.

Last but not least, I thank to my dog Čenda for his unending optimism.

Abstract

Thanks to numerous large-scale sequencing projects, the number of discovered genomic variants is increasing. The key step in analyzing the variant data is the functional annotation, since it helps researchers and clinicians to categorize, filter and prioritize the variants for further research. This thesis discusses five commonly-used variant consequence predictors, offers advice on how to use them and briefly goes through the algorithms they employ. Moreover, various data formats as well as the human reference genome and different genome annotations are described in the thesis. The correctness of the reference is of great importance as all the predictors rely on it. This thesis highlights some situations in which the results given by different predictors can vary. All the tests were made using the Ensembl gene annotation (release 92) and the GRCh38 reference assembly.

Keywords: variant consequence predictors, functional annotation, ANNOVAR, VEP, Haplosaurus, BCFtools/csq, SnpEff, predictors comparison

Abstrakt

Díky mnohým rozsáhlým sekvenačním projektům se množství nalezených genomických variant stále zvyšuje. Klíčovým krokem v analýze těchto dat je jejich funkční anotace, jež pomáhá varianty kategorizovat, filtrovat a prioritizovat pro další výzkum. Tato práce seznamuje s pěti běžně používanými programy pro určování důsledků variant, poskytuje rady, jak je používat, a stručně představuje algoritmy, které používají. Mimo to jsou zde popsány různé datové formáty, genomové anotace a lidský referenční genom. Správnost reference je velice důležitá, neboť na ní spoléhají všechny programy. Práce upozorňuje na určité situace, ve kterých se výsledky z různých programů mohou navzájem lišit. Pro všechny testy byla použita Ensembl genová anotace (release 92) a referenční genom GRCh38.

Klíčová slova: programy pro určování důsledků variant, funkční anotace, ANNOVAR, VEP, Haplosaurus, BCFTools/csq, SnpEff, srovnání programů

Contents

1. Introduction

1.1 What are genomic variants?

'Reference genome' is a collection of nucleotide sequences representing genomic information of a species (see chapter 2 for more information). Nucleotide sequences present in an individual typically differ from the reference at many sites; these are called 'genomic variants'. According to the 1000 Genomes Project [1], a typical human genome differs from the reference genome at 4.1 million to 5 million sites.

We distinguish many types of variants. They can be small-scale or large-scale and can have very different impact on the phenotype.

Single nucleotide polymorphisms (SNPs) are, as the name suggests, changes in only one nucleotide. We distinguish transitions (purine changes to purine or pyrimidine to pyrimidine) and transversions (purine to pyrimidine or vice versa). These small-scale variations are the most common variants in the human genome; 84.7 million out of 88 million variants identified in the 1000 Genomes Project were SNPs [1]. Although SNPs are very small, they can hit genes and cause changes in the protein sequence. Moreover, they can be located at important sites such as promoters or splice sites and most importantly, they can result in premature stop codon. Therefore, they are an important source of variation among individuals [2].

So called 'indel' is a common term for both insertions (adding one or more bases into the sequence) and deletions (deleting one or multiple bases). More than 99.9 % of all variants comprise of SNPs and short indels [1].

Structural variations affect larger regions of the genome and they are typically formed by rearrangements of the genome. A CNV (copy number variant) is defined as '*a DNA segment that is 1 kb or larger and present at variable copy number in comparison with a reference genome*' by Redon *et al.* [3]. Although structural variants are not as common (an estimated number is 2,100 to 2,500 per genome), they can affect ∼20 million bases of human genome sequence [1].

A 'non-synonymous' variant results in a codon change which alters the protein sequence. If the codon changes but encodes the same amino acid, the variant is called 'synonymous'. A 'nonsense' mutation results in a premature stop codon. There are also 'stop-loss' and 'frameshift' variants, which usually have high impact on the protein sequence. Frameshift means that the open (translational) reading frame was disrupted by an indel such that the number of inserted or deleted bases was not a multiple of three [4].

1.2 Variant consequence prediction

Thanks to extensive large-scale whole-exome and whole-genome sequencing projects such as 1000 Genomes [1], NHLBI-ESP [5] or UK10K [6], the amount of produced variation data is increasing. However, the exact functional impacts of most of the discovered variants remain unknown and manual examination of such a huge amount of data would be practically impossible; therefore, computational approaches are needed.

Variant annotation is a key step in sequencing data analysis. It is a process of assigning functional information to the variants [7]. Knowing the funcional consequences allows scientists and clinicians to categorize, filter and prioritize the variants for further analysis, for example to discover potential disease-causing mutations, to find new drug targets, to identify cancer driver mutations or to use the information in evolutionary studies. Incorrect predictions can result in an important disease-related variant being

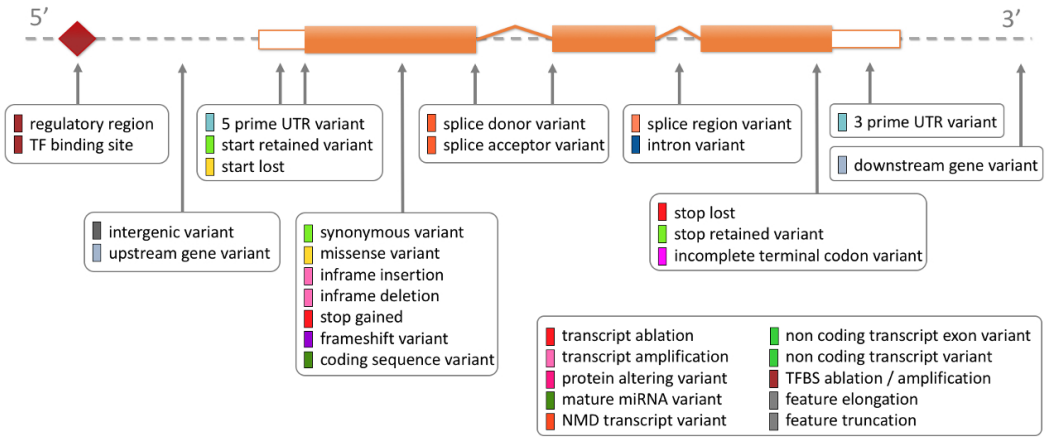

Figure 1.1: Consequence terms used by the Ensembl database and the Variant Effect Predictor. The diagram shows the locations of consequences in the transcript structure. Adapted from the *Ensemble database*. Retrieved 20-07-2018 from `https://www.ensembl.org/info/genome/variation/prediction/predicted_data.html`

overlooked or a harmless variant being marked as deleterious. [8, 9].

A fundamental step in the annotation process is to decide which part of the transcript the variant hits. Figure 1.1 shows the locations of consequences relative to the transcript structure. In general, we are more interested in the coding sequence (CDS) variants, since they can alter the amino acid sequence. Nevertheless, other parts of transcript are still important, for example for splicing or regulation, and they should be well-annotated as well [7].

If the variant hits a coding transcript, it is further analyzed which part of the transcript it overlaps. It can be an intronic or a UTR variant; in that case it is usually not inspected further. If the variant is exonic, the coding effects such as synonymous or missence amino acid replacement, frameshifts or stop codon gaining and loss are analyzed.

Variations in splice sites are usually very severe and it is therefore important to recognize them. Most tools regard variants in the first two bases of the intron as 'splice donor' variants and those in the last two bases as 'splice acceptor' variants. However, sometimes it is possible to change the number of bases on the boundaries that will be taken into account (this is the case of ANNOVAR [10] or SnpEff [11]). Most tools also consider 'splice region' variants. Those are defined as '*a sequence variants in which a change has occured either within 1-3 bases of the exon or 3-8 bases of the intron*', according to the Sequence Ontology [12].

1.2.1 Consequence description

Each tool has its list of terms used for the functional consequence description. Different tools can recognize different types of consequences. To give an example, ANNOVAR [10] uses the term *splicing* for both *splice_acceptor_variant* and *splice_donor_variant* used by the VEP [13], but it does not consider *splice_region_variant*. On the other hand, it distinguishes between frameshift insertions, deletions and block substitutions, while other tools report only *frameshift*.

It is standard to use the Sequence Ontology terms for describing the consequences. The Sequence Ontology (SO) [12] is a structured vocabulary and provides a consistent standardized set of terms and relationships. It simplifies the exchange of information

among different sources. For instance, the term CDS is clearly defined as '*a contiguous RNA sequence which begins with, and includes, a start codon and ends with, and includes, a stop codon*', and thus there is no need to discuss whether the stop codon is a part of the CDS or the 3' UTR. The advantage is that the terms are very computer-friendly; they contain underscores instead of spaces and the numbers and symbols are spelled out in most cases [12].

The terms are ordered hierarchically. All the terms describing the variants are under the *sequence_variant* term (see figure 1.2).

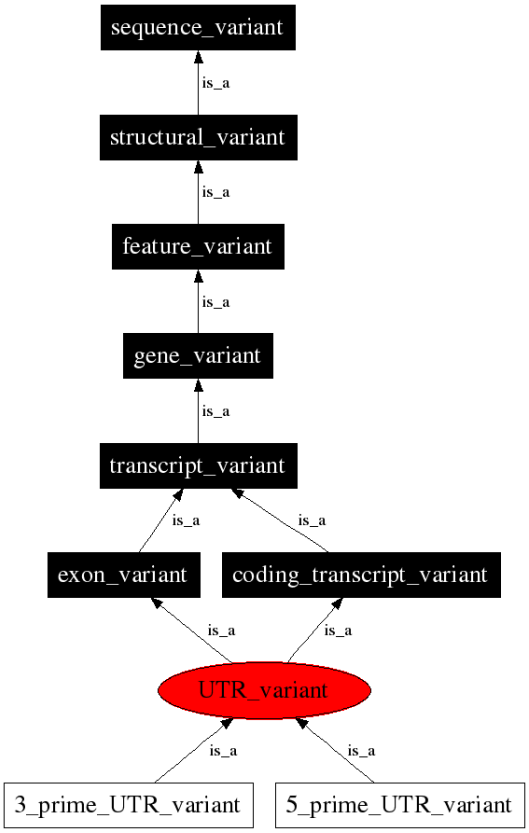

Figure 1.2: Relationship tree for the Sequence Ontology term *UTR_variant* (*SO:0001622*). Adapted from the *Sequence Ontology browser*. Retrieved 18-07-2018 from `http://www.sequenceontology.org/browser/current_svn/term/SO:0001622`

1.3 Limitations of predictors

Most of the commonly used variant consequence predictors (such as VEP [13], SnpEff [11], ANNOVAR [10], VAAST 2.0 [14]) can only analyze variants separately and they do not take into account phased haplotypes data, which is commonly accessible these days. Not considering the compound effect of more variants can cause incorrect predictions (see figure 1.3). The so called '*haplotype-aware consequence calling*' can solve the problem [15]. There are some tools that can handle the haplotype information, for instance BCFtools/csq [15], COPE [16] or Haplosaurus [17]. Selected predictors are described in chapter 4.

In some cases the variant consequence can be unambiguous. Genes often have more different transcripts (isoforms) and besides that, genes can overlap at a given position in

the genome. Therefore, it is not uncommon that the variant overlaps more transcripts and thus has more consequences. Software tools solve this complication differently; they can prioritize the consequences and report the most severe one or they return multiple annotations (for each overlapped transcript). However, typical pipelines are not prepared to handle a single variant with more consequences [7].

The variant prediction strongly depends on the genome annotation. Each annotation has (more or less) a different set of transcripts; it is therefore unsurprising that the predicted consequences can vary when using different gene annotations. Indeed, this statement was confirmed by McCarthy *et al.* in their research [7]. However, it is not clear-cut what the "best" annotation is.

For all those reasons, the variant consequence prediction can sometimes be complex and poses significant challenges.

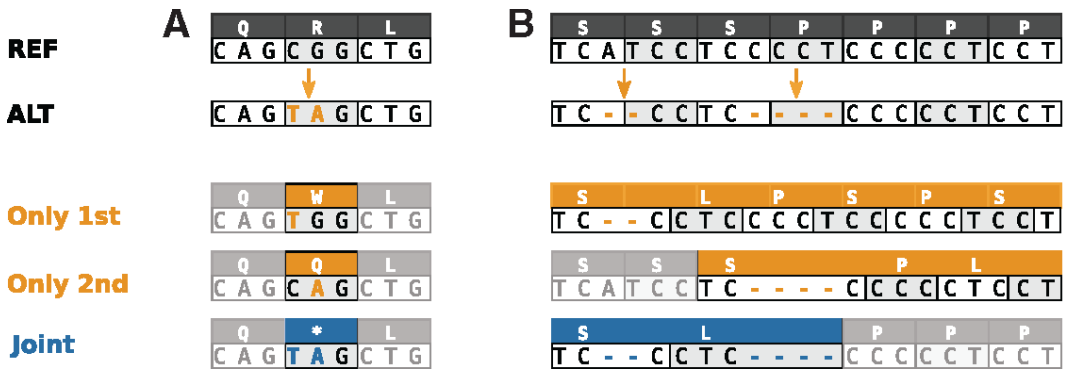

Figure 1.3: Incorrect predictions when not considering the compound effect. (**A**) Two SNPs together result in a stop codon. (**B**) A frameshift restoring variant. The AAs change is much less severe when considering the compound effect. Adapted from *BCFtools/csq: haplotype-aware variant consequences* [15].

2. Genome data representation

2.1 Reference genome

The term 'genome' can be defined in several ways in biology; in the following text, 'genome' stands for the genetic material present in a cell of a certain species. In case of the human, the haploid genome consists of 22 autosomal chromosomes, plus X and Y chromosomes and mitochondrial DNA.

The chromosomes are represented as linear nucleotide sequences. The most commonly used methods to sequence a genome are high-throughput sequencing methods (also known as NGS). However, despite rapid development of NGS techniques, it is not possible to obtain a sequence of the whole chromosome all at once. The sequences must be fragmented and shorter (hundreds of bases long) fragments are then sequenced separately; we obtain so called 'reads'. They are put together into contiguous longer sequences called 'contigs', that are ordered and cover the chromosome sequence [18].

The reference genome sequence can be understood as a 'template' representing genomic nucleotide sequence of a species. For most species including human, it is not a sequence of a single individual; it is rather a 'mosaic' created from sequences of many genomes to represent the biggest possible number of genes, transcripts and proteins present in population.

The reference is used in many genomics projects; its up-to-dateness and accuracy is therefore highly important [19].

2.1.1 Human reference genome

The Human Genome Project (HGP) [20] was established in 1990 and its goal was to create a complete and accurate human genome sequence. In 2001, the International Human Genome Sequencing Consortium (IHGSC) provided a draft sequence [21], which had certain drawbacks. There were approximatelly 150,000 gaps in the sequence (in highly repetitive regions of the genome, centromeres and telomeres) and about 10 % of euchromatic genome was omitted [22].

In 2007, The Genome Reference Consortium [23] was founded with purpose of improving and maintaining the reference genome sequences of selected species (currently human, mouse, zebrafish and chicken). The Consortium consists of five major bioinformatics institutions:

- Wellcome Sanger Institute [24]
- The McDonnell Genome Institute at Washington University (MGI) [25]
- The European Bioinformatics Institute (EBI) [26]
- The National Center for Biotechnology Information (NCBI) [27]
- The Zebrafish Model Organism Database [28]

Genome assembly

To represent more complex parts of the genome, a robust model was introduced by GRC; this model is called 'genome assembly model' [29]. It is able to represent so called 'alternative loci'. To fully represent some regions, it may be necessary to produce more than one sequence; this can be the case of large-scale structural variations or regions with high population diversity, such as the MHC region [30].

So far, there were two so called 'major releases' of the reference human genome assembly; the first one in February 2009 (GRCh37) and the second one (GRCh38) in December 2013. In addition to that, there are more frequent (quarterly for human) 'minor releases' which does not change genomic coordinates, but release 'patches', contig sequences meant to add information to the assembly. There are two types of patches; 'fix' patches correcting errors in the assembly, and 'novel' patches representing new alternate loci. [31]

The latest human genome assembly is GRCh38.p12 (patch 12) released in December 2017, adding 70 fix and 70 novel patches. Next update is planned for summer 2018. Thanks to incessant and long-lasting work on the human genome reference sequence, is is now of a high-quality. Nevertheless, there are still 875 gaps in the current version of the assembly [23]. Those are represented with letter 'N' in chromosomal sequence.

All released assemblies are publicly and freely available on the GRC website [23] and can be displayed in a 'friendly' graphical way in various genome browsers (UCSC [32], Ensembl [33]).

2.2 Genome annotations

The aim of the genome annotation is to identify locations of key genomic features, such as genes, transcripts, coding regions or regions important for regulation, and thus to create a kind of a 'genome map'. This annotation is the fundamental step in interpretation of the genome reference sequences, and as more and more scientists rely on it, the quality assurance is of great importance [34, 35].

Genome annotations are provided using different methods and resulting in data sets that are similar but certainly not the same. For high-quality finished genomes, such as the human or the mouse genome, manual annotation is needed to obtain high-accuracy data sets [35].

2.2.1 RefSeq

Reference Sequence (RefSeq) [36] database is a curated collection of linked genome, transript and protein sequence records built and maintained by NCBI. Records are derived from the data available in the GenBank database [37], but in contrast with GenBank, the RefSeq database is non-redundant. The goal is to maintain a set of stable, well-annotated and quality checked records; those records may thus contain additional information integrated from multiple resources, including functional features annotation, cross-references or informative nomenclature [36, 38].

For the purpose of annotation and quality evaluation, a significant number of tests is applied to all of the RefSeq records. Those quality assurance tests are designed to identify possible annotation problems like single exon genes, invalid stop and start codons, stop codons in CDSs, low-complexity sequences, CDSs shorter than 100 AAs etc. Test failure does not always mean that the record is necessarily incorrect; there are for instance some CDSs shorter than 100 AAs. However, they help to prioritize records for manual curation [38].

Every record has its accesion number which consists of a prefix followed by 6 or 9 numbers and a version number. The prefix indicates the type of the feature (M for mRNA, R for RNA, P for protein) and tells whether it is a model RefSeq record (letter X; generated through the annotation pipeline and not manually curated) or a known RefSeq record (letter N; curated). For instance, prefix *NM_* refers to a known protein-coding transcript [36]. More on RefSeq accession format, pipelines and other information can be found in The NCBI Handbook [39].

RefSeq is made publicly available and can be accessed via Entrez, BLAST, MapViewer or other NCBI tools. All data can be downloaded via the FTP protocol from the NCBI website [40]. A comprehensive FTP release is provided every two months, while updates and new records are released daily [41].

2.2.2 Ensembl gene annotation

Ensembl [42] provides gene annotations for selected vertebrate genomes using the Ensembl Gene Annotation system. This system is based on the alignment of biological sequences, such as cDNA, known protein sequences, or ESTs (expressed sequence tags) and the whole process is automatized, and thus it provides a fast way to annotate vertebrate genomes [43].

For some species, namely human, mouse, rat, zebrafish and pig, the Ensembl gene set is merged with HAVANA manual curration [44] to produce the final gene set. In case of human and mouse genome, the terms 'Ensembl annotation' and 'GENCODE annotation' are reffering to the same gene set [35, 45]. GENCODE annotation is described below.

Annotated features are assigned a stable Ensembl identifier. The ID comprises an '*ENS*' prefix followed by a species prefix, feature type prefix and 11 numbers. For example, '*ENSMUSG*' prefix reffers to a mouse gene. Complete list of Ensembl prefixes can be found at `https://www.ensembl.org/info/genome/stable_ids/prefixes.html`. The ID is followed by a dot and a version number which increases when a change in the feature happens [46].

Final gene sets and source code for the Ensembl Gene Annotation system are publicly and freely available on the Ensembl website. New releases are provided approximately every three months. Data can be downloaded via the Ensembl FTP site (`ftp://ftp.ensembl.org/pub/`) or accessed programatically through the Perl API [47] or the REST server [48]. [35]

2.2.3 GENCODE

GENCODE project aims to provide a highly accurate annotation of the human and the mouse genome. The process of annotation is very complex and includes automatic computational analysis by the Ensembl annotation system, manual annotation by the HAVANA group [44] and experimental validation (for example using RT-PCR-seq) [49], [50].

Every locus and transcript has a status assigned; possible values are 'known', 'novel' and 'putative'. 'Known' loci are represented in the HGNC database [51] and the RefSeq database [38], whereas 'novel' and 'putative' are not. 'Novel' loci are well supported by evidence while 'putative' loci have less extensive evidence [49].

The GENCODE data can be accessed in the UCSC genome browser [32]; it is also the default data set used in the Ensembl database [42]. Current and archived releases can be downloaded directly from the GENCODE website [52] using the FTP protocol. There are two main data sets — the Comprehensive gene annotation, containing all the transcripts, and the Basic gene annotation, which is a subset of the Comprehensive set and contains only full-length protein-coding transcripts [45].

3. Data formats

3.1 Generic Feature Format Version 3

The GFF3 format allows representation of genomic features in a readable, easily understandable and processable way, in contrast to relational database models. It is commonly used for the gene annotations. It is a tab-delimited format with 9 columns, where every line represents one genomic feature:

- **seqid** – ID of an object for establishing the coordinate system in which the current feature is located (for example chromosome number of name)
- **source** – source that generated the feature. Typically a database, a project or a software
- **type** – type of the feature described by a Sequence Ontology term [12] or an accession number
- **start** – start coordinate of the feature relative to the object in the first column.
- **end** – end coordinate of the feature relative to the object in the first column.
- **score** – score of the feature, i.e. E-value for sequence similarity features
- **strand** – orientation of the genomic feature, which can be either on the forward strand (+) or the reverse strand (-) (forward strand is the strand of the reference sequence).
- **phase** – it is required for features of type 'CDS' to set the reading frame. Phase '0' indicates that the reading frame starts at the same position as the feature starts; for phase '1', the first base is skipped and the first codon starts at the second position of the feature. For phase '2', the first two bases are skipped. For CDSs on the antisence strand, the phase is counted from the end
- **attributes** – semicolon-separated list of attributes described as *tag=value*. The list of possible attribute tags is available at `https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md`

If a field is undefined for a feature, a dot is filled in instead. Nonetheless, at least the *seqid, type, start* and *end* fields should be always defined.

Complete format specification can be found in the Sequence Ontology GitHub repositary [53].

3.2 FASTA

FASTA is a simple text format used for representation of nucleotide or amino acid sequences. There can be more sequences (i.e. all chromosomes of a genome) included in one file; every record begins with a one-row description introduced by a '>' symbol, followed by the sequence itself. Empty lines are not allowed in a FASTA file. There is not a strict specification for the first (description) line format, and thus different programs or institutions can have different requirements (if it is obligatory, identifiers usage, etc.) [54, 55].

3.3 Fai index

The Fai index format `faidx` enables faster and more effective access to FASTA files. The index file is a text file where every row contains a description of a FASTA sequence in the corresponding FASTA file. The tab-delimited columns are:

- sequence **ID**
- sequence **length**
- **offset** within the corresponding FASTA file (number of characters to the sequence first base)
- number of **bases on a row**
- number of **bytes on a row**, including the new line

Obviously, it is necessary to have the FASTA file accuratelly formatted according to the given description in the index file [56].

3.4 Human Genom Variation Society notation

In 2000, the Human Genome Variation Society (HGVS) [57] proposed a stable, consistent and comprehensive nomenclature for the genomic variants description, which has been accepted internationally.

All variants are described relatively to the reference sequence; the definition is meaningless without the reference. Therefore, every variant should be described as *reference:variant*, for example *NM_004006.2:c.3G>T*.

It is hard to say which reference sequences should be used and it remains an object of discussion. It is possible to use the genomic sequence, which is easy to work with - for example, there are not problems with alternative transcripts. However, a coding DNA reference sequence is often preferred in practice, because it is much easier to tell where in the CDS the variant is located (intron, exon, UTR, stop codon etc.). The HGVS recommends usage of a Locus Reference Genomic sequence (LRG) [58] or a RefSeq sequence [36].

Variants are described as *Prefix.PositionChange*. For instance, *c.1524G>A* means substitution of G to A at position 1524, reffering to the given reference. The prefix informs about the reference sequence: 'c' is used for coding DNA, 'g' for genomic DNA, 'n' for non-coding DNA, 'm' for mitochondrial DNA, 'r' for RNA and 'p' for protein. If the variant concerns more bases, both start and end positions are given, separated by an underscore. For an insertion, positions of bases surrounding the inserted bases are given; to give an example, *c.1485_1486insCCTG* is the insertion of four nucleotides between bases at positions 1485 and 1486 [59].

Detailed specification of more complex, large-scale structural variants is available on the HGVS website: `http://varnomen.hgvs.org/`.

3.5 Variant Call Format

Variant Call Format (VCF) is a text format used for storage and description of genomic variants. VCF was developed for the 1000 Genomes Project [1], but it was further used by other large-scale projects [60] such as UK10K [6], dbSNP [61] or NHLBI Exome Project [5]. Nowadays it is the standard for the variants description, and although it was developed for human variations, its flexibility exceeds the original intent.

It is possible to store millions of variants for thousands of samples (individuals) in a single VCF file. The important advantage is the possibility of representation of complex structural variants, such as large indels, CNVs, tandem duplications, transposonal sequence insertions, inversions etc.

VCF is divided into three parts - a meta-information section, a header line and variant records, each on a separate line.

Meta-information lines are prefixed with a '##' and must be in format *key=value*. These lines contain information about the VCF version, the reference sequence, the file creation date etc. Moreover, it can define format of the data lines in more detail, for instance tags for the genotype data description. Filters can be defined in this section as well; data lines that comply with the filter are skipped [62].

The header line is prefixed with a '#' and it names the 8 obligatory columns:

- **CHROM** – chromosome ID according to the reference genome
- **POS** – reference allele position in the reference sequence. Variants should be sorted by their position in increasing order
- **ID** – unique ID (for example dbSNP identifier)
- **REF** – reference allele
- **ALT** – alternate allele or a comma separated list of alternate alleles
- **QUAL** – 'Phred-scaled quality score' telling the ALT allele quality
- **FILTER** – semicolon-separated list of codes of filters that failed. If the record passed, the 'PASS' value is filled in
- **INFO** – additional information about the record. A semicolon separated list where each field is in the 'key=value' format

If the genotype information is present, there is the 9th 'FORMAT' column plus as many columns as there are samples. The 'FORMAT' field specifies format of the genotype information. It is a colon-separated list of keys such as 'GT' (genotype), 'DP' (read depth), 'HQ' (haplotype quality) or 'GQ' (genotype quality). The 'GT' field is compulsory and it tells what alleles are present in the sample. A '0' indicates the reference allele, '1' means the first alternate allele etc. For diploid calls there are two numbers separated by a '|' symbol for phased genotypes (where we can distinguish which of the two homologous chromosomes the each allele belongs to) or a '/' for unphased genotypes.

All the fields in the data lines are tab-separated and their order and count must correspond to the header line format. The missing values are represented by a dot [62].

The problem of the VCF file is its inefficiency for large amounts of variants in large-scale projects. It is a text file and therefore takes a lot of space (hundreds of GB) and its processing can be very slow. For this reason, there is BCF (Binary Variant Call Format) and its processing is much more efficient [60, 62].

3.6 Ensembl default

The default format used for the genomic variants description in the Ensembl database [42]. It is in the whitespace-separated format with five compulsory columns; the sixth column is not obligatory:

- **chromosone** number or name
- **start** position of the variant
- **end** position of the variant
- list of **alleles** separated by a '/'; the reference allele is the first one
- **strand**; '+' for forward and '-' for reverse strand
- variant **identifier**; when not provided, it is constructed from given coordinates and alleles [63]

3.7 Annovar input

This format is required as an input by ANNOVAR, the variant consequence predictor described hereinafter. It is a simple text format with every line describing a genomic variant with five whitespace-delimited fields:

- **chromosome** number
- **start** position of the reference allele
- **end** position of the reference allele
- **reference allele**
- **alternative allele**

It is possible to add additional information in other columns. A dash indicates missing allele (in case of insertion or deletion) and '0' can be given instead of the reference allele, if the information is not available. This format can represent insertions, deletions and substitutions (SNPs or block) [10, 64].

4. Genomic variants consequence predictors

4.1 Ensembl Variant Effect Predictor

The Variant Effect Predictor (VEP) is a tool provided by Ensembl [42]. It is used internally in the Ensembl database for annotation of newly-imported variants, and additionally, it can be used for variant consequence prediction, variant analysis and prioritization of user-supplied data. It supports annotation of most types of variant types in both coding and non-coding regions.

The main drawback is that it annotates every variant separatelly, not considering the compound effect. Nonetheless, a new tool called Haplosaurus was introduced in 2016 and it can process whole-transcript haplotype sequences. The tool is described in the following section. The VEP is open-source and free to use and it is actively maintained and further developed [13].

4.1.1 Usage

The most user-friendly and easiest way to learn working with the VEP is the VEP Web, an online tool available at `https://www.ensembl.org/Tools/VEP`. The user fills in the form, enters the variant data and alters various options, submits the query and waits until the job is finished. The VEP Web is more suitable for less experienced users and smaller analysis [65].

To make full use of the VEPs functionality, the best way is to use the VEP Perl script. Download and installation instructions are available at `http://www.ensembl.org/info/docs/tools/vep/script/index.html`. The script works the most efficiently in "offline mode", using a local cache of annotation files (Ensembl annotation [35] or RefSeq [36], both available for GRCh38 and GRCh37). Those can be downloaded automatically when running the installer script. More options and settings can be used compared to the VEP Web. Moreover, the input file size is completelly unlimited (for the VEP Web, the limit is 50 MB which is around two million lines in VCF file) [13].

Another way to use the VEP is via the REST API [48] which is accessible from any programming language. It returns basic variant annotations in JSON format which is simple for parsing.

4.1.2 Input

The input can be in various formats; it can be a list of variant IDs or descriptions in HGVS notation. Another option is to provide a VCF file or the Ensembl default formatted file. Those formats can represent selected structural variations; recognized values are 'INS', 'DEL', 'DUP' and 'TDUP' [63].

4.1.3 Output

The first output file is a HTML file with statistics and summary (number of overlapped transcripts, filtered out variants count, percentage of variants in the non-coding regions etc.). The second output file is in the TSV format by default; other possibilities are JSON, GVF [66] or VCF (annotation is in the 'INFO' column).

The consequence for each alternative allele and each overlapped genomic feature is written on a separate row [65].

Variant consequences are described by the Sequence Ontology terms [12]. If the amino acid sequence is modified by the variant, the VEP is able to provide pathogenicity predictor scores such as SIFT [67] or PolyPhen-2 [68].

The output files in VCF or TSV format can be filtered out using the `filter_vep` script. It can filter out known variants, variants in non-coding regions, variant with SIFT score less than 0.1, not located in the first exon etc. [65]. Detailed instructions for filtering the results are available at `https://www.ensembl.org/info/docs/tools/vep/script/vep_filter.html`.

4.1.4 Algorithm

The VEP algorithm code is part of the Ensembl API which is written in Perl and depends on the BioPerl API [69]. Time-critical parts are programmed in the C programming language. Comprehensive documentation of the Ensembl API is freely available [70].

First of all, the input file is processed; the Ensembl default format can be directly converted into a `VariationFeature` object. For variants in the HGVS notation, the genomic coordinates based on the reference sequence must be resolved. VCF is preprocessed, because the VCF and the Ensembl default format representation of the variants can differ. For instance, VCF requires to add one base before an indel to the alleles. The variant position is therefore shifted by one position compared to the Ensembl default.

The input variants are read into a memory buffer which is thereafter processed by more subprocesses, each having its own part of the data. All the results are merged together and sorted before returning the output.

Every variant goes through a quality-control process. For example, it checks whether the alleles match the coordinates, or whether the reference allele matches the reference genome sequence. Incorrect variants are not processed.

The genomic loci overlapped by the variants are separated to 1 Mb regions and a memory cache with transcripts and regulatory features is created for each region. Therefore, it does not have to be loaded repeatedly for each variant. For each transcript, the information about intron-exon structure, UTR, coding regions and translated regions is cached in the same manner later in the process.

A `VariationFeatureOverlap` object represents an overlap between an input variant and a genomic feature. A particular sub-class of that class is created for each overlap that was found; `TranscriptVariation`, `RegulatoryFeatureVariation` and `MotifFeatureVariation` are the possible subclasses.

A `TranscriptVariationAllele` (a `VariationFeatureOverlapAllele` subclass) object is a child class of the `TranscriptVariation`, representing an allele of the variant. For each `TranscriptVariationAllele` object, the consequences are evaluated using a set of predicate functions. These functions are built to decide certain conclusions about the variant. To give an example of a predicate: "*Does this variant change the protein coding sequence?*" If the answer is 'True', an `OverlapConsequence` object representing the consequence type is assigned to the `TranscriptVariationAllele`. One variant can have more `OverlapConsequence` objects assigned and because of that, the consequence with the highest priority can be selected at the end.

In purpose of speeding up the computation, there are also pre-predicate checks deciding which predicates need to be computed. For instance, we do not need to compute the amino acid sequence change for an intron variant.

Computed `VariationFeatureOverlapAllele` objects with predicted consequences are then processed for output. The filters (according to given input parameters) are

applied in this place. The plugin modules are executed at this stage as well. Therefore, the plugins work with the prepared output, but can alter it before it is written to the file. Most plugins modify only information in the last column ('Extra') of the output; however, it is possible to modify the output line in any manner. The `Bio::EnsEMBL::Variation::Utils::BaseVepPlugin` modul is recommended to implement new plugins [13, 70].

4.2 Haplosaurus

Haplosaurus is a tool provided by Ensembl [42] and can be downloadabed at `http://www.ensembl.org/info/docs/tools/vep/script/index.html` together with VEP. The advantage of Haplosaurus is that it is able to compute the compound effect of more variants and predict consequences for haplotypes.

The tool is implemented in the Ensembl database in the transcript haplotypes view [71]. In this view, we see a list of haplotypes originated from the 1000 Genomes Project and their relative frequency in population. A haplotype is expressed as a list of variations.

4.2.1 Usage

The `haplo` script is a command line tool and it shares most of the functionality with the VEP; most arguments are the same and it is possible to use the same local cache of the gene annotations.

Haplotype frequencies observed in the 1000 Genomes Project can be assigned using the `--haplotype_frequencies` flag [17].

4.2.2 Input

The only supported input format is a phased VCF file with data for at least one sample [17].

4.2.3 Output

The default format is a tab-delimited file, however, it is possible to get a JSON output using `--json`. The fields are as follows:

- **transcript ID**
- haplotype **name in HGVS notation** representing differences to the reference
- **flags for CDS** haplotype
- **protein name** in HGVS notation
- **flags for protein** haplotype
- **frequency data**
- contributing **variants**
- **sample:count** that exhibits this haplotype

4.2.4 Algorithm

A pair of haplotype sequences is created for each transript overlapping the variant. The haplotypes are constructed according to the phased genotype data in the VCF file. The haplotype sequences are translated to the protein sequences and compared to the reference [17].

4.3 ANNOVAR

ANNOVAR is another command line tool for genomic variants annotation. The most important functionality is so called *gene-based annotation*; it determines which genes the variant overlaps and it decides the consequences for each transcript. Besides, it is possible to perform *filter-based annotation* to look through a variation database (i.e. dbSNP [61] or 1000 Genomes Project data CITACE [1]) and determine if the variant is present there. Moreover, *filter-based annotation* can be performed to find important genomic regions (predicted transcript factors binding site, conserved regions, conserved RNA secondary structures etc.) overlapping with the variant [10].

4.3.1 Usage

The usage of ANNOVAR is simple; all the scripts are downloadable from `http://annovar.openbioinformatics.org/en/latest/user-guide/download/` and it is free to use for non-commercial purposes. The scripts are written in Perl and they are accessible from the command line without any installation. It only needs to download database files with reference sequences and annotations; the `annotate_variation.pl` script with the `--downdb` argument should be used for this purpose.

The most important functionality is the `annotate_variation.pl` script. We can choose the type of the annotation (`--geneanno, --regionanno, --filter`), the type of reference data (refGene, ensGene) and their location on the disc, the version of the reference genome assembly (hg18 (GRCh37) is the default) and the input file with variants.

Another possibility is to use the `table_annovar.pl` script. Its output is a tab-delimited file with detailed variant annotation. For example, it is possible to get SIFT [67] or PolyPhen-2 [68] score or a genetic disorder associated with the gene. Another advantage is that it is able to perform more types of annotations at the same time or to use different gene annotations [10]. The script internally uses `annotate_variation.pl`, so the results should be the same; the main advantage is having more annotations in one file together.

4.3.2 Input

The `annotate_variation.pl` script can process only the ANNOVAR input format which means we need to convert other types of formats to this default one. There is the `convert2annovar.pl` script that does the work for us; it can convert several formats including VCF.

When using the `table_annovar.pl` script, a VCF file can be provided with the `--vcfinput` argument. Nevertheless, the script calls the same conversion script stated above.

Unlike VCF, the ANNOVAR input format cannot represent more complex variants, but only indels and substitutions. Having more samples in one VCF file is problematical as well; only the data for the first sample are written to the converted file. We can change this behaviour by the `--allsample` argument; in that case a separate file is created for each sample [64].

4.3.3 Output

The gene-based annotation returns two files as an output, named after the input file with extensions `.variant_function` and `.exonic_variant_function`.

The first file contains a line for every variant in the input file. Two columns are added at the beginning of the input line. The first column tells which part of the transcript the variant hits (exonic, splicing, UTR5, downstream, intergenic,...). Only the term with the highest priority is stated; that can be changed by the `--separate` argument. The second column contains the name of the overlapped gene, or alternatively a comma-separated list of genes. If no gene is hit, then the two neighboring genes and their distance is stated.

The second output file annotates only variants marked as 'exonic'. In the first column, there is the variant line number. The second column tells the functional consequences. The third column contains the gene name, the transcript ID and described change of the transcript (with `--hgvs` argument it will be given in the HGVS nomenclature [59]).

The `coding_change.pl` script is needed to get the modified amino acid sequence. [64]

4.3.4 Algorithm

Firstly, the count of lines in the input file determines whether multithreading will be used. According to the autors observations, multithreading is beneficial if there is more than one million variants.

The type of the annotation is decided by the given argument and the corresponding function is called: `annotateQueryByGeneThread`, `annotateQueryByRegionThread`, or `filterQueryThread`. If multithreading is allowed, the input variants are distributed among the threads.

Therefore, the `annotateQueryByGeneThread` is called for the gene annotation. Firstly, the genome annotation files are parsed, then the input is parsed and checked for correctness by the `detectInvalidInput` function and the `processNextQueryBatchByGeneThread` processes a block of variants. For each variant it is decided whether it is intra- or intergenic and whether it is an upstream or downstream variant. Moreover, for variants inside genes it is determined if it hits an intron, an exon, a UTR or a splice site.

The `annotateExonicVariantsThread` is called for exonic variants. The FASTA sequence is loaded and, according to the amino acid sequence modification, it is decided whether the reading frame changed, whether a stop codon was gained or lost or whether it was a synonymous or nonsynonymous mutation [72].

4.4 BCFtools/csq

BCFtools is a set of utilities for variant calling and handling the VCF and BCF files. The `BCFtools/csq` command runs a fast and efficient haplotype-aware consequence predictor which can make use of known phased haplotype data and predict effects of compound variants. The program is written in the C programming language and it is very fast and efficient [15, 73].

Compared to the VEP or SnpEff, it does not offer such a rich functionality (such as reporting known variants, predicting pathogenicity scores etc.), but it is more focused on correct classification of nearby variants in known phase and interpretation of their compound effect.

There is also a possibility to run localized predictions with only one variant at a time when using the `--local-csq` option [73].

4.4.1 Usage

The BCFtools package download and installation instructions can be found at `http://www.htslib.org/download/`. The package internally uses HTSlib [74] which is distributed as a separate package or together with the BCFtools package and needs to be installed before installing BCFtools.

There is no need to build any database cache. Instead, the user supplies a reference genome sequence in the FASTA format (using `-f` or `--fasta-ref` options), a genome annotation in the GFF3 format (`-g, --gff-annot`) and a VCF file with input variants.

4.4.2 Input

The only possible input format is the VCF (or BCF). It should contain phased haplotype information; nevertheless, if the phase is unknown, the `--phase` option can be used to determine how to handle unphased heterozygous genotypes [73].

4.4.3 Output

The predicted effects are written to the 'INFO' field (the eight column) of the input file using the 'BCFQ' tag. A comma-separated list of overlapped transcript annotations in the following format is given:

- **Consequence** type
- **Gene** name
- Ensembl **transcript** ID
- **Biotype**
- **Strand** (+/-)
- **Amino acids change**
- **DNA change** (list of corresponding variants)

The annotations for variants downstream to a stop codon are prefixed with a '$*$'.

If the variants have compound effect, one of them has the full annotation assigned and the other ones are referenced with the position of the annotated one. For instance,

```
BCSQ=missense|CLASP1|ENST00000545861|-|1174P>1174L|
|122106101G>A+122106102G>A
BCSQ=@122106101
```

In this example, two variants (at positions 122106101 and 122106102) change the same codon. The second annotation gives the reference to the first one instead of the full annotation.

There can be many samples in the VCF file which means there are also different haplotypes and it makes the representation of the consequences more complicated. Consequences for each haplotype are written to the 'FORMAT' field for each sample as a bitmask of indexes. The bitmask can be translated into a readable format using the `BCFtools/query` command. For instance, the command

```
bcftools query -f'[%CHROM\t%POS\t%SAMPLE\t%TBCSQ\n]' csqOutput.vcf
```

prints consequences for all the haplotypes in separate columns. For more information see the BCFtools manual [73].

4.4.4 Algorithm

The first thing is parsing the gene annotations in the GFF3 file. Each transcript has a gene assigned as a parent and all the CDSs, exons and UTRs have the corresponding transcripts assigned. Then the search for overlapping regions (CDSs, UTRs, exons or general transcripts) is performed.

Overlapped transcripts are kept on a heap data structure. A haplotype tree is built for each transcript according to the genotype information in the phased VCF file. The nodes correspond to the VCF records and each node has as many child nodes as there are alleles in the record. Therefore, the leaves of the tree represent different haplotypes and the internal nodes represent haplotypes with the same 'prefix' shared by multiple samples.

When all the variants in the transcript are processed, the transcript is spliced and the consequences are decided [15].

4.5 SnpEff

SnpEff (an abbreviation of 'SNP effect') is a variant annotation and effect prediction tool, which can annotate thousands of variants per second. The tool is open source and freely available. The code is written in Java and it is platform independent. Beside the functional effect prediction, it supports many annotations, such as loss of function (LOF) and nonsense-mediated decay (NMD) predictions or assigning the SIFT [67] or PolyPhen-2 [68] scores. It produces variant annotations in HGVS notation [59]. SnpEff can also perform non-coding and regulatory annotations, but the corresponding databases must be available.

4.5.1 Usage

The download and installation are very easy. The ZIP file can be downloaded from `http://snpeff.sourceforge.net/download.html` and all that is needed to do is to unzip the file.

SnpEff requires a database to produce the annotations. The database data are downloaded and installed automatically when doing predictions for the first time and there is no need to do it manually in most cases. However, it is possible to build custom databases from supplied GFF/GTF and FASTA files in case there is not a pre-build one that would suit the users needs. Detailed instructions for building the databases are available in the online documentation [75].

4.5.2 Input

VCF is the strongly recommended input format, since it is a standard format used by other software packages [11].

4.5.3 Output

SnpEff supports TXT and VCF output formats. However, VCF is the default one and it is highly recommended to use it.

The annotations are added to the 'INFO' field (the eight column) of the VCF file using the 'ANN' tag. If there are more genes or transcripts affected by the variant, and therefore there is more than one annotation, all of them are reported, separated by commas.

Each annotation consists of 16 sub-fields separated by a '|'. The sub-fields are:

- **alternative allele**
- predicted **effect** – Sequence Ontology terms by default
- **impact** – HIGH, MODERATE, LOW or MODIFIER
- **gene name**
- **gene ID**
- **feature type** (transcript, motif, miRNA, etc.)
- **feature ID**
- transcript **biotype** (coding, non-coding i.e.)
- exon (intron) rank / total number of exons (introns)
- variant described in **HGVS notation**
- **protein change** (if any) in HGVS notation
- position in cDNA / cDNA length
- position in CDS / CDS length
- position in protein sequence / protein sequence length
- **distance to feature** (i.e. for upstream variants, distance to closest gene)
- errors, warnings and information

The consequences are described by the SO terms [12] by default.

A file with summary and statistics (number of known variants, transitions/transversions ratio, percentage of exon variants etc.) is created as well. Since calculating the statistics can take a lot of time, it is recommended to disable it by the `-nostats` argument when it is not needed [11].

The output can be modified by filters. Some of them are pre-implemented, but custom output filters can be supplied as well, using SnpSift filters [76]. SnpSift is a toolbox that can be downloaded together with SnpEff and allows to manipulate the annotated files.

4.5.4 Algorithm

Firstly, SnpEff needs to load the binary database stored by SnpEff as compressed serialized objects. It takes some time but after it is loaded, SnpEff is very effective.

When loading a database, SnpEff builds a data structure which can, given any interval or point, efficiently find all overlapping intervals. For each contig in the genome assembly, an 'interval tree' is built. It is a binary tree data structure where each node stores a center point and all intervals overlapping the center, sorted by beginning and end positions. The node has a pointer to another node containing all intervals completely to the left of the center, and another one for those to the right. Moreover, to reduce the number of intervals, a hash is built of those interval trees, indexed by chromosomes; this data structure is called an 'interval forest' [11].

The genome statistics are calculated at this point. After building the data structure for efficient interval search, the input file is parsed and the overlapping genomic regions are found for each region. If those regions include an exon, the coding consequences are calculated.

SnpEff supports multithreading, but it is not used by default and it must be switched on by the `-t` argument. Size for splice sites is 2 by default and the default upstream and downstream size is set to 5kb, but these settings can be changed [11].

5. Practical part – Comparison of the predictors

5.1 Objectives of the practical part

- to create a set of tests (variants represented in VCF files) that can help to test predictors in various situations
- to discuss the results of the tests and compare annotations given by five selected variant effect predictors
- to highlight situations in which the prediction of consequences can be problematic

5.2 Methods

Using the Ensembl genome browser (release 92), a transcript sequence was manually examined to create test cases with variants located in various parts of the transcript and causing diverse consequences. Emphasis was put on problematic regions such as exon-intron boundary, CDS-UTR boundary or stop and start codons. Compound variants were tested as well.

The annotations were made with the Ensembl transcripts set (release 92, 05-04-2018) and the GRCh38.p12 reference genome assembly. Some variants were encoutered in real data and some were made-up to cover various cases that could possibly happen. Five different variant consequence predictors were used for annotations:

- VEP, v92.5
- Haplosaurus, v92.5
- ANNOVAR, version 2018Apr16
- BCFtools/csq, bcftools-1.9 + htslib-1.9
- SnpEff, version 4.3T

Installation of the programs was made according to the documentations. For VEP and Haplosaurus, the database cache was created automatically together with the installation, using the

```
ftp://ftp.ensembl.org/pub/release-92/variation/VEP/
  /homo_sapiens_vep_92_GRCh38.tar.gz
```

file from the Ensembl FTP site. The GFF3 file and FASTA reference sequence for BCFtools/csq were also downloaded from the Ensembl FTP site:

```
ftp://ftp.ensembl.org/pub/release-92/fasta/homo_sapiens/dna/
  /Homo_sapiens.GRCh38.dna.chromosome.22.fa.gz
```

```
ftp://ftp.ensembl.org/pub/release-92/gff3/homo_sapiens/
  /Homo_sapiens.GRCh38.92.chromosome.22.gff3.gz
```

SnpEff builds the binary database automatically when doing the predictions for the first time. It downloaded the files from

```
http://downloads.sourceforge.net/project/snpeff/
  /databases/v4_3/snpEff_v4_3_GRCh38.92.zip
```

ANNOVAR downloads the files from the UCSC Table Browser [32]. Following the instructions in the documentation, the GENCODE V28 Comprehensive data set (`wgEncodeGencodeCompV28` UCSC table) was downloaded using the `annotate_variation.pl` script with the `--downdb` option. GENCODE V28 (version 28, 05-04-2018) corresponds to the Ensembl 92 annotations.

The downstream and upstream regions were taken as 5 kb regions adjacent to the transcription start site and transcription end site. The size 5 kb was set by default in VEP and SnpEff. The default for ANNOVAR is 1 kb, but this was changed by the `--neargene` option. BCFtools/csq and Haplosaurus do not annotate upstream and downstream regions. The splice site size was set to 2 bases by default for all the tools.

Following commands in listings 5.1 – 5.2 were used for the annotations:

```
# The annotate_variation.pl script cannot process VCF input. We need to
    convert it first:
perl "$annovarPath"/convert2annovar.pl --format vcf4 "$vcfFilePath" >
    avinput.txt
# We are performing gene-based annotation (--geneanno) with the Ensembl
    gene annotation and hg38 (GRCh38) reference genome.
# All functional consequences (rather than just the most important one)
    are printed out when using the --separate option.
perl "$annovarPath"/annotate_variation.pl --geneanno --dbtype ensGene
    --buildver hg38 --neargene 5000 --separate avinput.txt
    "$annovarPath"/humandb/
```

Listing 5.1: ANNOVAR

```
# The variant processing is much faster with --cache and --offline. The
    cache can be downloaded throught the installer.
# --vcf causes that the output is written in VCF format
# --regulatory allows looking for overlaps with regulatory features
# --no_stats disables generating a statistics file
perl "$vepPath"/vep --cache --offline --vcf --input_file "$vcfFilePath"
    --regulatory --species homo_sapiens --no_stats
```

Listing 5.2: VEP

```
# The variant processing is much faster with --cache and --offline. The
    cache can be downloaded throught the installer.
perl "$vepPath"/haplo --cache --offline --input_file "$vcfFilePath"
    --species homo_sapiens
```

Listing 5.3: Haplosaurus

```
"$bcftoolsPath"/bcftools csq -f "$FASTAPath" -g "$GFF3Path"
    "$vcfFilePath"
```

Listing 5.4: BCFtools/csq

```
# The java parameter -Xmx4g is used to increase the memory available to
    the Java Virtual Machine to 4G.
# To perform regulatory annotations, a regulatory database needs to be
    build. The instructions are available in the documentation
    ("Building databases: Regulatory and Non-coding"). The -reg option
    specifies the regulation track to use.
# --nostats disables generating a statistics file
java -Xmx4g -jar "$snpeffPath"/snpEff.jar GRCh38.92 "$vcfFilePath" -reg
    Predicted_promoter -nostats
```

Listing 5.5: SnpEff

5.3 Results

105 VCF test files were made and all of them were annotated by the predictors. The results were compared with the expected consequences and a table with comparisons and commentaries was created. The table and the VCF files are available on the attached CD.

The predictors all worked well for uncomplicated variants, such as intergenic variants, intron variants, UTR variants, missense and synonymous SNPs, frameshifts and inframe indels. Intergenic, upstream and downstream variants are not reported by BCFtools/csq for its low importance. Haplosaurus does not annotate any variants outside the CDSs. Only VEP and SnpEff can do regulatory and non-coding annotations (ANNOVAR can find overlapped regulatory regions when performing the region-based annotation).

There are differences in consequence terms used by different predictors. Both VEP and SnpEff use the Sequence Ontology terms. BCFtools/csq uses SO as well, but omits the word 'variant' (so it uses for example *intron* instead of *intron_variant*). ANNOVAR and Haplosaurus have their own sets of terms. The latter often reports only the change in the transcript/protein sequence instead of a consequence term.

There can be differences in the types of consequences that are recognized. For example, SnpEff is the only one that reports the *exon_loss_variant* and only VEP reports the *NMD_transcript_variant*. ANNOVAR does not consider the *splice_region_variant* at all. Haplosaurus uses the *stop_change* term for both *stop_gained* and *stop_lost* variants. ANNOVAR does not distinguish between *splice_donor_variant* and *splice_acceptor_variant* and uses the term *splicing* for both of them. ANNOVAR reports *wholegene* when the whole start codon was deleted (but not if there is only a change in the start codon). There are even more differences and the lists of the used terms can be found in the programs documentations.

Handling structural variants is much more complex and can be very limited when working with the discussed tools. It is recommended to rather use a specialized tool, such as AnnotSV [77]. ANNOVAR can identify previously reported structural variants when doing the region-based annotation. BCFtools/csq and Haplosaurus skip structural variants. VEP and SnpEff can annotate basic structural variants. VEP currently recognizes four values written in the "SVTYPE" INFO field in VCF: INS(insertion), DEL(deletion), DUP(duplication) and TDUP(tandem duplication).

Next sections describe some of the encountered problems.

Stop codon gained

Some tools had problems with the *stop_gained* variant and reported frameshift instead. An example is pictured in figure 5.1.

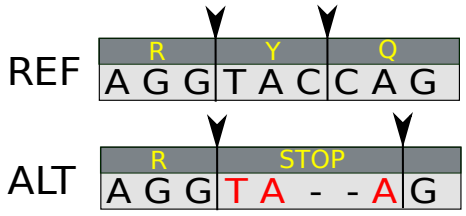

Figure 5.1: 2 bp deletion in the `stop_gained_deletion` test creates a new stop codon. All the tools except for VEP returned *frameshift_variant* instead of *stop_gained*.

Stop codon retained

There can be situations when the stop codon is changed, but it is either changed into a different stop codon, or it is shifted because of an indel. The tools should return the *stop_retained_variant*, or the *synonymous SNV* variant in case of ANNOVAR. This consequence is much less severe than the *stop_lost*.

The tools work well in the simple cases, such as a SNP in a stop codon (see test `stop_retained_SNP`). Nevertheless, it gets complicated even with simple indels. In the example in figure 5.2, the situation is not so complex, but only VEP seems to return the right output (see table 5.1)

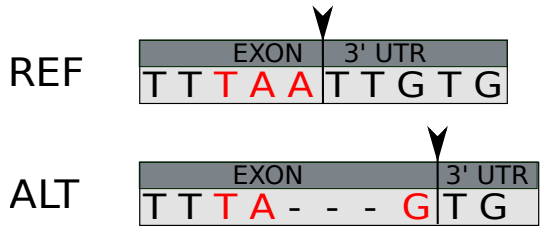

Figure 5.2: 3 bp deletion in the `stop_retained_deletion` test changes the stop codon, but a new one is created. Stop codons are highlighted in both sequences.

ANNOVAR	frameshift deletion
VEP	stop_retained_variant & 3_prime_UTR_variant
SnpEff	stop_lost & conservatice_inframe_deletion & 3_prime_UTR_variant
BCFtools/csq	stop_lost & 3_prime_utr
Haplosaurus	XXX

Table 5.1: The `stop_retained_deletion` test results. Only VEP recognized the retained stop codon.

The example in the figure 5.3 shows situation where only BCFtools/csq and VEP correctly identify the *stop_retained_variant*, but in addition, all the five predictors report *frameshift*, which is incorrect because the translation ends with the newly created stop codon.

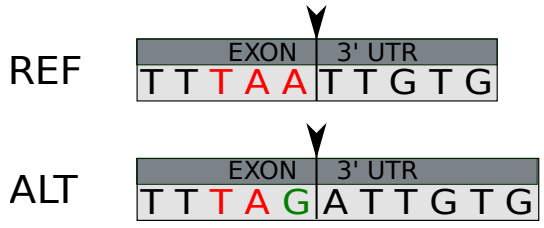

Figure 5.3: The `stop_retained_insertion` test. The insertion of 'G' creates new stop codon. No frameshift.

In the `stop_lost_insertion` test, BCFtools/csq returned wrong output - *stop_retained* instead of *stop_lost*.

Whole exon deletion

It is hard to say what consequence should be reported when the whole exon is deleted. The best option seems to be the *exon_loss_variant*, because it describes the situa-

tion the best, but it is only used by SnpEff. However, even SnpEff does not report *exon_loss_variant* when the first or the last codon is deleted. The reason could be that the *stop_lost* and *start_lost* variants have higher priority (see the `last_exon_deletion` and `first_exon_deletion` tests). In the `last_exon_deletion` test, ANNOVAR should be able to report *stoploss*.

Retained splice sites and ambiguous representation of variants in border regions

In the example pictured in figure 5.4, the same alternate sequence was obtained by insertions at two different sites. The resulting sequence is the same and thus the predicted consequences should be the same for both variants.

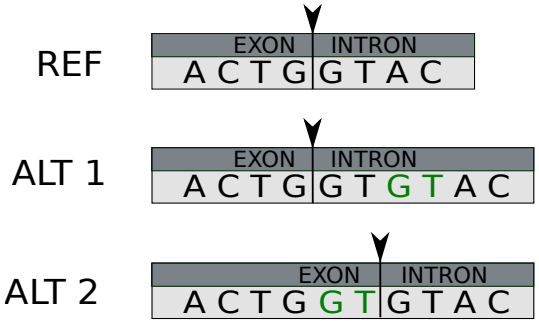

Figure 5.4: The same alternate sequence was obtained in two tests: **(a)** ALT 1: `splice_donor_retained_insertion_2` and **(b)** ALT 2: `splice_region_insertion`. The inserted bases are highlighted.

However, the results differ, because the variant is treated as a splice region variant in the first case and as a CDS variant in the second one (see table 5.2). Only SnpEff seems to treat both variants equally.

	ALT 1	**ALT 2**
ANNOVAR	splicing & intronic	frameshift insertion
VEP	splice_region_variant & intron_variant	frameshift_variant & splice_region_variant
SnpEff	splice_region_variant & intron_variant	splice_region_variant & intron_variant
BCFtools/csq	splice_region	synonymous & splice_donor
Haplosaurus	XXX	frameshift & indel

Table 5.2: Different consequences predicted for the same alternate sequence.

However, it is hard to say which consequence should be expected. It is likely that the splicing would not be disrupted, as the splice site technically did not change. In that case, frameshift variant would be the most suitable. However, splicing is a complicated process and a change in sequence near splice site could cause some problems. Therefore, the information about the splice region change should not be completelly omitted on the output. Maybe the *splice_region_variant* & *frameshift_variant* would be the best option in this case.

Similar situations can be seen in other pairs of tests:

- `splice_acceptor_retained_1` and `splice_acceptor_retained_2`
- `splice_acceptor_retained_3` and `splice_acceptor_retained_4`

If a *splice_site_retained* term existed, it would be an easy solution. The consequence would be less severe than *splice_acceptor* and *splice_donor* variants, but different from the *splice_region_variant*.

It seems that SnpEff takes the retained splice site sequences into notice and reports *splice_region_variant* instead. See the `splice_donor_retained_insertion` and `splice_donor_retained_deletion` tests as an example.

Start codon

Start codon loss is very severe, because the translation cannot start at the site and the protein sequence will be changed. ANNOVAR does not report the *start_lost* consequence at all. It uses *wholegene*, but only if the whole start codon was deleted.

If there is a change in the stop codon, but it is retained, the *start_retained* consequence should be reported. However, it seems that only VEP should be able to use this term and it did not use it correctly in any test. All the predictors except for SnpEff seem to have problem in this situation. As an example, see figure 5.5.

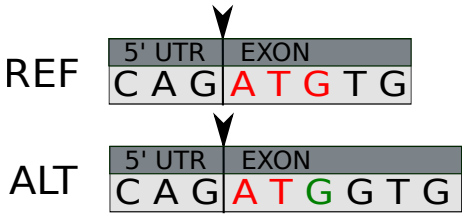

Figure 5.5: The `start_codon_retained_insertion_1B` test. An insertion of 'G' does not change the start codon. All the predictors correctly report *frameshift*. However, VEP and BCFtools also report *start_lost* variant, which is incorrect.

An interesting situation, when the start codon changes into a stop codon, is pictured in figure 5.6. The predicted consequences can be found in table 5.3. ANNOVAR incorrectly predicted the *stopgain* variant, because the translation cannot start, so the stop codon is unimportant. Other tools correctly reported the *start_lost* variant, but reported *frameshift* as well, which would be odd in this case, as we do not know where the translation starts.

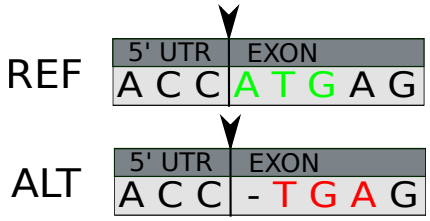

Figure 5.6: The `start_to_stop_deletion` test. The start codon changed into a stop codon.

ANNOVAR	stopgain
VEP	frameshift_variant & start_lost
SnpEff	frameshift_variant & start_lost
BCFtools/csq	frameshift & start_lost
Haplosaurus	XXX

Table 5.3: The `start_to_stop_deletion` test results.

Effects of compound variants

Out of the five tools, only BCFtools/csq and Haplosaurus can decide the consequences of compound variants. While BCFtools/csq handles most types of consequences, Haplosaurus is intended for the haplotypes only, and thus it resolves only variants in CDSs and ignores intergenic, intronic or splice site variants. However, it does not handle even variants overlapping the boundaries (exon-intron, exon-UTR) and it can result in missing annotation of the CDS. For instance, a stop codon loss is not reported because the variant goes beyond the boundary (see the `stop_lost_deletion` test). Another example is the whole exon deletion; it certainly concerns the CDS, but Haplosaurus does not annotate it. See the `middle_exon_deletion` test - the whole exon is deleted, but the splice sites are not changed, and Haplosaurus still does not annotate this variant.

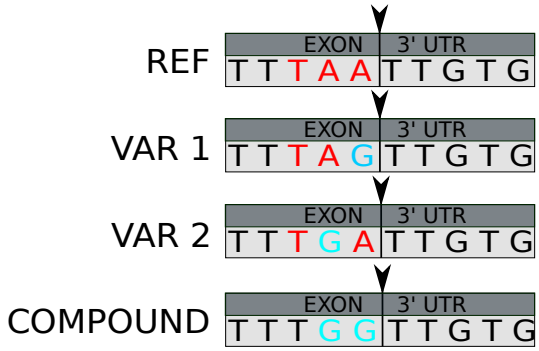

Figure 5.7: The `SNP+SNP_stop_loss` test. Two SNP variants change the stop codon and are annotated as *stop_retained_variant* when treated separately. Nevertheless, the correct consequence is the stop codon loss. The compound variant is correctly handled by BCFtools/csq and Haplosaurus.

Haplosaurus was correct in all the tests with compound variants. BCFtools/csq was correct in most cases, but there were two tests in which the wrong consequence was returned. The first one was the `indel+indel_splice_region_exonic_frame_restored` test. A four base deletion together with a two base deletion should result in a frame restoring variant. For unknown reason, BCFtools/csq treated the variants separately and the resulting consequence was frameshift. Both deletions were located in exonic splice regions; maybe this could be the issue, because otherwise BCFtools/csq can recognize the restored frameshift variant without problem (as an example, see `indel+indel_different_exon_frame_restored`). Another incorrect prediction given by BCFtools/csq can be seen in figure 5.8.

Figure 5.8: The `indel+indel_start_codon_retained_frame_restored` test. Two deletions result in a 9 base frame restoring deletion and the start codon retains. BCFtools returns *frameshift & start_lost* variant. The reason is that it thinks the variants overlap and it skips the second deletion.

Conclusion

Variant consequence predictors have an important role in variant annotation pipelines and are of great help in many studies. However, the computational variant consequence prediction is not as simple as it seems. The existing variant consequence predictors are not faultless in all situations and the results should not be trusted blindly, as the predictions can sometimes be incorrect or incomplete. A possible solution is to use different tools and compare the results. In some cases (usually for variants located on the exon-intron boundaries, exon-UTR boundaries or in stop and start codons), the correct consequence is not obvious. These variants could be reported as problematic on the output and they could be further examined manually. This solution might be better than having an incorrect prediction which causes an important variant not to be noticed. When working with haplotype data, a tool that is able to handle compound variants should be used, since the differences in the predictions can greatly differ. As a follow-up to this thesis, it could be useful to create even larger set of test cases to cover all the situations that could possibly happen, and to invent an algorithm that would solve the problems encountered in this thesis.

List of Abbreviations

AA	Amino acid
API	Application Programming Interface
BLAST	Basic Local Alignment Search Tool
CDS	Coding sequence
CNV	Copy number variation
COPE	Context-Oriented Predictor
dbSNP	Database for Single Nucleotide Polymorphisms
EBI	European Bioinformatics Institute
ENCODE	Encyclopedia of DNA elements
EST	Expressed sequence tag
FASTA	Fast alignment
FTP	File Transfer Protocol
GFF3	Generic Feature Format Version 3
GRC	Genome Reference Consortium
GTF	Gene Transfer Format
GVF	Genome Variation Format
HAVANA	Human and Vertebrate Analysis and Annotation
HGNC	HUGO Gene Nomenclature Committee
HGP	Human Genome Project
HGVS	Human Genome Variation Society
ID	Identifier
IHGSC	International Human Genome Sequencing Consortium
JSON	JavaScript Object Notation
LOF	Loss of function
LRG	Locus Reference Genomic sequence
MGI	McDonnell Genome Institute at Washington University
MHC	Major histocompatibility complex
NCBI	National Center for Biotechlonogy Information
NGS	Next generation sequencing
NHLBI	National Heart, Lung and Blood Institute
NHLBI-ESP	NHLBI Exome Sequencing Project
NMD	Nonsense-mediated decay
PolyPhen	Polymorphism Phenotyping
RefSeq	Reference Sequence
REST	Representational State Transfer
RT-PCR	Reverse transcription polymerase chain reaction
SIFT	Scale-invariant feature transform
SNP	Single nucleotide polymorphism
SnpEff	SNP effect
SO	Sequence ontology
TSV	Tab-Separated Values
TXT	Text file
UCSC	University of California, Santa Cruz
UTR	Untranslated region
VAAST	Variant Annotation, Analysis and Search Tool
VCF	Variant Call Format
VEP	Variant Effect Predictor

Bibliography

[1] 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, October 2015.

[2] Ryan E. Mills, Christopher T. Luttig, et al. An initial map of insertion and deletion (INDEL) variation in the human genome. *Genome research*, 16:1182–1190, September 2006.

[3] Richard Redon, Shumpei Ishikawa, Karen R. Fitch, Lars Feuk, George H. Perry, T. Daniel Andrews, Heike Fiegler, Michael H. Shapero, Andrew R. Carson, Wenwei Chen, et al. Global variation in copy number in the human genome. *Nature*, 444(7118):444, 2006.

[4] The Sequence Ontology Browser. "`http://www.sequenceontology.org/browser/`", 2016. Accessed: 15-07-2018.

[5] Exome Variant Server, NHLBI GO Exome Sequencing Project (ESP). "`http://evs.gs.washington.edu/EVS/`". Accessed: 12-07-2018.

[6] UK10K Consortium et al. The UK10K project identifies rare variants in health and disease. *Nature*, 526:82–90, October 2015.

[7] Davis J. McCarthy, Peter Humburg, Alexander Kanapin, Manuel A. Rivas, Kyle Gaulton, Jean-Baptiste Cazier, and Peter Donnelly. Choice of transcripts and software has a large effect on variant annotation. *Genome Medicine*, 6(3):26, 2014.

[8] William McLaren, Bethan Pritchard, Daniel Rios, Yuan Chen, Paul Flicek, and Fiona Cunningham. Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor. *Bioinformatics (Oxford, England)*, 26:2069–2070, August 2010.

[9] S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, M. R. Speicher, J. Zschocke, and Z. Trajanoski. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefings in Bioinformatics*, 15(2):256–278, January 2014.

[10] Kai Wang, Mingyao Li, and Hakon Hakonarson. ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. *Nucleic acids research*, 38:e164, September 2010.

[11] Pablo Cingolani, Adrian Platts, Le Lily Wang, Melissa Coon, Tung Nguyen, Luan Wang, Susan J. Land, Xiangyi Lu, and Douglas M. Ruden. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3. *Fly*, 6:80–92, 2012.

[12] Karen Eilbeck, Suzanna E. Lewis, Christopher J. Mungall, Mark Yandell, Lincoln Stein, Richard Durbin, and Michael Ashburner. The Sequence Ontology: a tool for the unification of genome annotations. *Genome biology*, 6:R44, 2005.

[13] William McLaren, Laurent Gil, Sarah E. Hunt, Harpreet Singh Riat, Graham R. S. Ritchie, Anja Thormann, Paul Flicek, and Fiona Cunningham. The Ensembl Variant Effect Predictor. *Genome biology*, 17:122, June 2016.

[14] Hao Hu, Chad D. Huff, Barry Moore, Steven Flygare, Martin G. Reese, and Mark Yandell. Vaast 2.0: improved variant classification and disease-gene identification using a conservation-controlled amino acid substitution matrix. *Genetic epidemiology*, 37:622–634, September 2013.

[15] Petr Danecek and Shane A. McCarthy. BCFtools/csq: haplotype-aware variant consequences. *Bioinformatics*, 33(13):2037–2039, feb 2017.

[16] Si-Jin Cheng, Fang-Yuan Shi, Huan Liu, Yang Ding, Shuai Jiang, Nan Liang, and Ge Gao. Accurately annotate compound effects of genetic variants using a context-sensitive framework. *Nucleic acids research*, 45:e82, June 2017.

[17] Ensembl. Haplosaurus documentation. "`https://www.ensembl.org/info/docs/tools/vep/haplo/index.html`". Accessed: 19-07-2018.

[18] Ansorge W. J. Next generation DNA sequencing (II): Techniques, applications. *Journal of Next Generation Sequencing & Applications*, 01(S1), 2015.

[19] Derek L. Stemple. So, you want to sequence a genome... *Genome Biology*, 14(7):128, 2013.

[20] Leroy Hood and Lee Rowen. The human genome project: big science transforms biology and medicine. *Genome medicine*, 5:79, 2013.

[21] International Human Genome Sequencing Consortium et al. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, February 2001.

[22] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431:931–945, October 2004.

[23] Genome Reference Consortium. "`https://www.ncbi.nlm.nih.gov/grc`". Accessed: 09-07-2018.

[24] Wellcome Sanger institute. "`https://www.sanger.ac.uk/`". Accessed: 11-07-2018.

[25] McDonnell Genome Institute at Washington University. "`http://genome.wustl.edu/`". Accessed: 18-07-2018.

[26] The European Bioinformatics Institute. "`https://www.ebi.ac.uk/`". Accessed: 18-07-2018.

[27] The National Center for Biotechnology Information. "`https://www.ncbi.nlm.nih.gov/`". Accessed: 18-07-2018.

[28] The Zebrafish Model Organism Database. "`http://zfin.org/`". Accessed: 18-07-2018.

[29] Valerie A. Schneider et al. Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome research*, 27:849–864, May 2017.

[30] NCBI Genome Assembly Model. "`https://www.ncbi.nlm.nih.gov/assembly/model/`". Accessed: 07-07-2018.

[31] Definitions - Genome Reference Consortium. "`https://www.ncbi.nlm.nih.gov/grc/help/definitions#ALTERNATE`". Accessed: 11-07-2018.

[32] Donna Karolchik, Angie S. Hinrichs, and W. James Kent. The UCSC Genome Browser. *Current protocols in bioinformatics*, Chapter 1:Unit1.4, December 2012.

[33] Victoria Newman, Benjamin Moore, Helen Sparrow, and Emily Perry. The Ensembl Genome Browser: Strategies for Accessing Eukaryotic Genome Data. *Methods in molecular biology*, 1757:115–139, 2018.

[34] Lincoln Stein. Genome annotation: from sequence to biology. *Nature Reviews Genetics*, 2(7):493–503, jul 2001.

[35] Bronwen L. Aken and Aothers. The Ensembl gene annotation system. *Database : the journal of biological databases and curation*, 2016, 2016.

[36] Nuala A O'Leary et al. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic acids research*, 44:D733–D745, January 2016.

[37] Dennis A. Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, David J. Lipman, James Ostell, and Eric W. Sayers. GenBank. *Nucleic acids research*, 45:D37–D42, January 2017.

[38] K. D. Pruitt, T. Tatusova, and D. R. Maglott. NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, 35(Database):D61–D65, jan 2007.

[39] Kim Pruitt, Garth Brown, Tatiana Tatusova, and Donna Maglott. The NCBI Handbook, Chapter18. "`https://www.ncbi.nlm.nih.gov/books/NBK21091/`", 2002. Accessed: 04-07-2018.

[40] RefSeq: NCBI Reference Sequence Database. "`https://www.ncbi.nlm.nih.gov/refseq/`". Accessed: 04-07-2018.

[41] Kim D. Pruitt, Garth R. Brown, Susan M. Hiatt, Françoise Thibaud-Nissen, Alexander Astashyn, Olga Ermolaeva, Catherine M. Farrell, Jennifer Hart, Melissa J. Landrum, Kelly M. McGarvey, Michael R. Murphy, Nuala A. O'Leary, Shashikant Pujar, Bhanu Rajput, Sanjida H. Rangwala, Lillian D. Riddick, Andrei Shkeda, Hanzhen Sun, Pamela Tamez, Raymond E. Tully, Craig Wallin, David Webb, Janet Weber, Wendy Wu, Michael DiCuccio, Paul Kitts, Donna R. Maglott, Terence D. Murphy, and James M. Ostell. RefSeq: an update on mammalian reference sequences. *Nucleic acids research*, 42:D756–D763, January 2014.

[42] Daniel R. Zerbino et al. Ensembl 2018. *Nucleic acids research*, 46:D754–D761, January 2018.

[43] Val Curwen, Eduardo Eyras, T. Daniel Andrews, Laura Clarke, Emmanuel Mongin, Steven M. J. Searle, and Michele Clamp. The Ensembl automatic gene annotation system. *Genome research*, 14:942–950, May 2004.

[44] John M. Hancock. HAVANA (Human and Vertebrate Analysis and Annotation), 2004.

[45] Adam Frankish, Barbara Uszczynska, Graham R. S. Ritchie, Jose M. Gonzalez, Dmitri Pervouchine, Robert Petryszak, Jonathan M. Mudge, Nuno Fonseca, Alvis Brazma, Roderic Guigo, et al. Comparison of GENCODE and RefSeq gene annotation and the impact of reference geneset on variant effect prediction. *BMC genomics*, 16(8):S2, 2015.

[46] EMBL-EBI. Ensembl genome browser 92, Stable IDs. "https://www.ensembl.org/info/genome/stable_ids/index.html". Accessed: 14-06-2018.

[47] EMBL-EBI. Ensembl Perl API Documentation. "http://www.ensembl.org/info/docs/api/index.html". Accessed: 07-07-2018.

[48] WTSI and EBI. Ensembl REST API. "https://rest.ensembl.org/". Accessed: 11-07-2018.

[49] Jennifer Harrow et al. GENCODE: the reference human genome annotation for The ENCODE Project. *Genome research*, 22:1760–1774, September 2012.

[50] Cédric Howald, Andrea Tanzer, Jacqueline Chrast, Felix Kokocinski, Thomas Derrien, Nathalie Walters, Jose M. Gonzalez, Adam Frankish, Bronwen L. Aken, Thibaut Hourlier, Jan-Hinnerk Vogel, Simon White, Stephen Searle, Jennifer Harrow, Tim J. Hubbard, Roderic Guigó, and Alexandre Reymond. Combining RT-PCR-seq and RNA-seq to catalog all genic elements encoded in the human genome. *Genome research*, 22:1698–1710, September 2012.

[51] Elspeth A. Bruford, Michael J. Lush, Mathew W. Wright, Tam P. Sneddon, Sue Povey, and Ewan Birney. The HGNC Database in 2008: a resource for the human genome. *Nucleic acids research*, 36:D445–D448, January 2008.

[52] GENCODE. "https://www.gencodegenes.org/". Accessed: 15-06-2018.

[53] Lincoln Stein. Generic Feature Format Version 3 (GFF3). Available online at https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md. Accessed: 10-03-2018.

[54] UniProt FASTA headers. "https://www.uniprot.org/help/fasta-headers". Accessed: 15-07-2018.

[55] NCBI. Blast documentation. "https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp". Accessed: 14-06-2018.

[56] SAMtools. Fai index description. "http://www.htslib.org/doc/faidx.html". Accessed: 14-03-2018.

[57] Human Genome Variation Society. "http://www.hgvs.org/". Accessed: 10-03-2018.

[58] Raymond Dalgleish et al. Locus reference genomic sequences: an improved basis for describing human DNA variants. *Genome medicine*, 2:24, April 2010.

[59] Johan T. den Dunnen, Raymond Dalgleish, Donna R. Maglott, Reece K. Hart, Marc S. Greenblatt, Jean McGowan-Jordan, Anne-Francoise Roux, Timothy Smith, Stylianos E. Antonarakis, and Peter E. M. Taschner. HGVS recommendations for the description of sequence variants: 2016 update. *Human mutation*, 37:564–569, June 2016.

[60] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The variant call format and VCFtools. *Bioinformatics (Oxford, England)*, 27:2156–2158, August 2011.

[61] S. T. Sherry, M. H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin. dbSNP: the NCBI database of genetic variation. *Nucleic acids research*, 29:308–311, January 2001.

[62] SAMtools. The Variant Call Format Specification , 2018. Available online at `https://samtools.github.io/hts-specs/VCFv4.3.pdf`. Accessed: 10-03-2018.

[63] EMBL-EBI. VEP Data formats. "`https://www.ensembl.org/info/docs/tools/vep/vep_formats.html#default`". Accessed: 02-04-2018.

[64] ANNOVAR Documentation. "`http://annovar.openbioinformatics.org/en/latest/`". Accessed: 10-04-2018.

[65] EMBL-EBI. Ensembl VEP documentation. "`https://www.ensembl.org/info/docs/tools/vep/index.html`". Accessed: 14-06-2018.

[66] Martin G. Reese, Barry Moore, Colin Batchelor, Fidel Salas, Fiona Cunningham, Gabor T. Marth, Lincoln Stein, Paul Flicek, Mark Yandell, and Karen Eilbeck. A standard variation file format for human genome sequences. *Genome biology*, 11:R88, 2010.

[67] Prateek Kumar, Steven Henikoff, and Pauline C. Ng. Predicting the effects of coding non-synonymous variants on protein function using the sift algorithm. *Nature protocols*, 4:1073–1081, 2009.

[68] Ivan Adzhubei, Daniel M. Jordan, and Shamil R. Sunyaev. Predicting functional effect of human missense mutations using PolyPhen-2. *Current protocols in human genetics*, Chapter 7:Unit7.20, January 2013.

[69] Jason E. Stajich, David Block, Kris Boulez, Steven E. Brenner, Stephen A. Chervitz, Chris Dagdigian, Georg Fuellen, James G. R. Gilbert, Ian Korf, Hilmar Lapp, Heikki Lehväslaiho, Chad Matsalla, Chris J. Mungall, Brian I. Osborne, Matthew R. Pocock, Peter Schattner, Martin Senger, Lincoln D. Stein, Elia Stupka, Mark D. Wilkinson, and Ewan Birney. The Bioperl toolkit: Perl modules for the life sciences. *Genome research*, 12:1611–1618, October 2002.

[70] EMBL-EBI. Ensembl Perl API Documentation. "`https://www.ensembl.org/info/docs/api/index.html`". Accessed: 19-07-2018.

[71] EMBL-EBI. Ensembl Transcript haplotype. "`http://www.ensembl.org/Help/View?id=564`". Accessed: 19-07-2018.

[72] Kai Wang, Mingyao Li, and Hakon Hakonarson. ANNOVAR code. Downloadable at: `http://annovar.openbioinformatics.org/en/latest/user-guide/download/`. Accessed: 20-06-2018.

[73] BCFtools manual page. "`https://samtools.github.io/bcftools/bcftools.html`". Accessed: 24-07-2018.

[74] Wellcome Sanger Institute. SAMtools / BCFtools / HTSlib. "`https://www.sanger.ac.uk/science/tools/samtools-bcftools-htslib`". Accessed: 24-07-2018.

[75] Pablo Cingolani. SnpEff documentation. "`http://snpeff.sourceforge.net/SnpEff_manual.html`". Accessed: 05-07-2018.

[76] Pablo Cingolani et al. Using Drosophila melanogaster as a model for genotoxic chemical mutational studies with a new program, SnpSift. *Frontiers in genetics*, 3:35, 2012.

[77] Véronique Geoffroy, Yvan Herenger, Arnaud Kress, Corinne Stoetzel, Amélie Piton, Hélène Dollfus, and Jean Muller. Annotsv: An integrated tool for structural variations annotation. *Bioinformatics (Oxford, England)*, April 2018.

A. Attachments

The attached CD contains two attachments:

A.1 VCF_files

Folder with all the VCF files that were used as test cases for comparing the annotations.

A.2 Annotation_results_table

Table with results of the tests and their comparison with the expected consequence.