Algorithms for the Description of Molecular Sequences

Jonathan K. Vis

This publication was supported by the Dutch national program COMMIT.

Typeset using LaTeX, figures generated using TIKZ
Printed by Ridderprint B.V.
ISBN 978-90-9030094-8

Algorithms for the Description of Molecular Sequences

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 21 december 2016
klokke 11:15 uur

door

Jonathan Klaas Vis
geboren te Leuven, België
in 1983

Promotiecommissie

Promotores	prof.dr. J.N. Kok	
	prof.dr. P.E. Slagboom	
Copromotor	dr. J.F.J. Laros	
Commissieleden	prof.dr.ir. F. Arbab	
	prof.dr. H.J. van den Herik	
	dr. S.D. Olabarriaga	Universiteit van Amsterdam
	prof.dr. A. Plaat	
	dr. P.E.M. Taschner	

Contents

Chapter 1

Introduction

Bioinformatics is a a broad research field operating on the boundary between bio-medical research and computer science and often mathematics and statistics. In particular the automated analysis of biological data and the for that purposes developed tools are prime research interests. The analysis of genomic data is one of its main focus points. In this dissertation we will address two topics in bioinformatics; the analysis of molecular sequences and the application of *machine learning* on large cohort studies. The chapters dealing with molecular sequences will mainly focus on the development of novel algorithms (and ultimately tools) which are useful for the automated analysis of those sequences. Many algorithms and tools have been developed in the past, e.g., aligners and variant callers. Here, we target a related subject; the generation of descriptions which are typically used within a clinical setting. The second part of this dissertation has a more explorative nature. By using existing tools and methods we explore the added classification power of adding disjunct groups of data to a study.

In Chapter 2 we first introduce the basic biological background knowledge needed when dealing with molecular sequences from a bioinformatics perspective. Next, the domain specific language that is used for describing variation within molecular sequences is introduced.

1.1 Outline

In this dissertation we explore a particular field within the larger bioinformatics field; we focus on algorithms for generating descriptions for molecular strings. The first chapters deal specifically with different aspects of the construction of Human Genome Variation Society Nomenclature (HGVS) descriptions.

The primary research question in Chapter 3 is: "Can we automatically generate concise, meaningful HGVS descriptions from DNA sequences in linear time?" This chapter focuses in the most general way on the construction of genomic description of DNA/RNA sequences. Here, we explore an algorithm that is able to efficiently compute these descriptions for very large strings, e.g., whole chromosomes. The challenge is to generate concise, yet meaningful, descriptions. This chapter is based on:

Vis, J. K., Vermaat, M., Taschner, P. E. M., Kok, J. N., and Laros, J. F. J. (2015). An Efficient Algorithm for the Extraction of HGVS Variant Descriptions from Sequence. *Bioinformatics*, 31(23):3751–3757.

In Chapter 4 the research question is: "Can we automatically generate HGVS descriptions from protein sequences as well?". This chapter builds on the algorithm presented in Chapter 3 and aims to construct descriptions for protein sequences. Its main focus is on the detection of so-called frameshift variants. In this we show that descriptions on DNA/RNA level are generally insufficient for describing variants on the protein level. And we show that interesting evolutionary events can be found in real-life data when looking at frameshift variants.

The research question in Chapter 5 is: "Can we efficiently generate descriptions for regions with repeated sequences?". We focus on finding so-called short tandem repeats and describing repeat structures with regard to some reference sequence in terms of those repeats. Regions with high content of repeats are common within the genome, at the same time they are difficult to analyze. A specialized application of the algorithm from Chapter 3 is presented for constructing accurate and meaningful repeat descriptions free from any database-based method. A particular focus is given to the application in

forensic research.

In Chapter 6 we have as primary research questions: “Can we say something about the relative power of the (combinations of) sets of attributes?” and: “Can data from different cohort studies be used to augment classifying power for a single study?”. We take our focus of molecular strings and we explore the problem of classification in large cohort studies containing heterogeneous data. This is in many aspects a study of machine learning on a meta level, where we investigate the added classification power of adding disjunct groups of data and disjunct cohorts. The data used in these cohorts often contains genomic data. The algorithms presented in the Chapters 3,4,5 could be applied in these kind of analyses. This chapter is based on:

Vis, J. K. and Kok, J. N. (2014). *Meta-analysis of Disjoint Sets of Attributes in Large Cohort Studies*, pages 407–419. Springer Berlin Heidelberg, Berlin, Heidelberg.

Finally, in Chapter 7 we summarize the conclusions of the individual chapters and we present directions for future work.

Chapter 2

Preliminaries

The core of every living being is its genetic payload. The genetic inheritance describes information that is passed down from parents to their offspring. It contains a blueprint detailing, in essence, how to construct a new individual from a single cell. This genetic inheritance is physically present in the form of *DNA* in almost every living cell.

As a medium of information storage, DNA is complemented by two other types of molecules in the cell that, respectively, carry out the instructions encoded in the DNA by performing specific biochemical functions, and serve as an intermediary between information storage and execution. The intermediaries, which are called *RNA*, copy out specific parts of the complete instructions from the DNA and carry them to factories that translate the instructions into *proteins*. The *central dogma of molecular biology* states that information is thus transmitted from DNA to RNA, and from RNA to proteins, but never from proteins back to RNA or DNA [Crick, 1958, Crick et al., 1970]. When first published, the central dogma concisely summarized the available evidence at the time. Now, more than half a century later, this still largely holds true.

Over the years, the very high-level view of the central dogma was complemented by a detailed mechanistic description and the efforts to fill in all the details are still ongoing. The three main roles in the central dogma are fulfilled by DNA, RNA and proteins, respectively, and we are now going to take a look at all of them in turn.

2.1 DNA

DNA consists of a long chain of *nucleotide* molecules consisting of a ribose, one or more phosphate groups and a *nucleobase*: adenine (A), cytosine (C), guanine (G) or thymine (T). Thus, DNA can be thought of as a long string of four different letters, and that is indeed how it is often represented. Text is written from left to right in Western cultures. By convention, DNA is written from $5'$ to $3'$.

DNA is present in the cell in the form of double-stranded helices: each DNA molecule consists of two paired chains, wound tightly around each other, with the bases on each chain pairing up such that every A on one chain is paired with a T on the other, and each C is paired with a G. This symmetry is known as Watson-Crick base pairing, after its discoverers [Watson et al., 1953]. Thus, DNA is made up of two complementary strands, redundantly holding the genetic information, see Figure 2.1. This redundancy is used in DNA copying, which occurs at every cell division, and is the mechanism by which genetic information is passed from one cell to its offspring, to synthesize two newly formed DNA molecules, each of which contains one strand of the parent DNA molecule [Meselson and Stahl, 1958].

DNA is not made up of one single polymer chain, but rather is partitioned into several long pieces, called *chromosomes*. Each chromosome forms a single molecule. However, even on a chromosome the genetic information is not stored in one consecutive piece: Rather, DNA consists of relatively short stretches encoding a specific function, separated by long stretches that do not directly encode any function. The "function" is what is transmitted, as per the central dogma, to RNA and, in many cases, on to proteins. Such self-contained, functional stretches are called *genes* self-contained stretch of DNA that is transcribed to perform a function. To perform its function, a gene has to be transcribed into RNA.

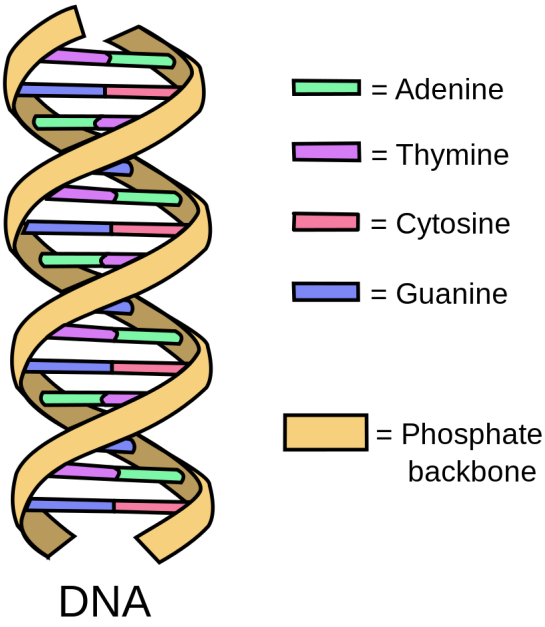


Figure 2.1: A schematic representation of the helical structure of double-stranded DNA. Also shown is the complementary nature of the nucleotides. Image by Forluvoft [Public domain], via Wikimedia Commons.

2.2 RNA

RNA is the product of transcription of a gene from DNA. RNA is an information carrier like DNA, but unlike the latter, RNA is created as a single strand. This has two consequences: First, RNA is much less stable than DNA, and slowly degrades. RNA thus has a finite life-time, and the pool of RNA must be replenished by continuous transcription. Second, single-stranded ribonucleic acid spontaneously changes its spatial conformation by forming Watson-Crick base pairs between nucleotides in its own sequence. The resulting structure, although not the only factor, can confer biochemical functions to the RNA. Because the structure is determined by, and exists on a higher level than the

sequence identity of the RNA, it is called *secondary structure*.

Another difference between DNA and RNA is the use of slightly different nucleobases: instead of `T`, RNA uses `U` (uracil), which, like `T`, base-pairs with `A`. Despite the fact that the genetic information is encoded in virtually the same way in DNA and RNA, transcription of DNA into RNA requires a complex machinery. The core of this machinery is a complex enzyme called an *RNA polymerase*. In eukaryotes, three different, evolutionarily related RNA polymerases are responsible for transcribing different types of RNA.

RNA performs numerous different functions, but one very important subcategory of RNA does not perform any function on its own; rather, it is an intermediary between the genetic information on the DNA and the final protein product, which in turn performs cellular functions. This class of RNA is called mRNA. mRNA is the product of the transcription of protein-coding genes. Transcription of mRNA requires an exquisite control, and many different transcription factors are known to regulate the activity of transcription of different genes in different cellular conditions.

This results in different mRNA genes being transcribed at highly different levels, leading to several orders of magnitude of difference in mRNA abundance. Moreover, the same mRNA can be transcribed at different levels under different conditions. This forms the basis of cellular differentiation into different cell types and tissues in multicellular eukaryotes.

2.3 Proteins

Proteins, finally, are the main effectors of cell function. Like DNA and RNA, they consist of chains of smaller molecules, so-called *amino acids*, that are strung together to form *polypeptides*. Each amino acid is a small molecule with unique properties which, jointly, dictate the function of the final protein. Individual amino acids are strung together in a chemical reaction to form a *peptide bond* [Alberts et al., 1995]. Polypeptides, like RNA, form secondary structures via non-covalent bonds between amino acids, which are a function of the amino acid sequence. Beyond this, proteins form even higher order three-dimensional structures called tertiary structures. When multiple proteins

aggregate into a complex consisting of several subunits this is called quarternary structure.

All these different levels of spatial organization of proteins lead to the creation of highly complex structures from originally one-dimensional chains. It is their intricate structure that allows them to perform precise tasks in the cell. Because they are the work horses of the cells, proteins are highly abundant, with some proteins being present million-fold at any given moment. This is only possible because a single gene is transcribed multiple times, and each resulting mRNA can be translated several times, and simultaneously, before being degraded. The path DNA $\rightarrow$ RNA $\rightarrow$ protein thus facilitates an amplification from a single gene copy to many orders of magnitudes more copies of the resulting protein. Despite the fact that multiple protein copies can be created from a single mRNA molecule, and that the number varies from transcript to transcript, protein abundance is predominantly determined by the abundance of mRNA.

2.4 Transcription

As mentioned previously, different polymerases are responsible for transcribing genes encoded in the DNA into different types of RNA. The precise ways in which the different polymerases transcribe genes into their RNA products differ but the fundamental aspects of transcription are similar. In all cases, a *motif* in the DNA sequence initiates binding of a number of transcription factor proteins to the DNA. Such motifs, called *promoters*, are found in the immediate vicinity of the transcription splice site of their target genes — either upstream of the transcription splice site or following closely after it, inside the gene body. Once the transcription factors have bound to the DNA on top of the transcription site, the RNA polymerase attaches to the DNA and is held in place by the transcription factors. Subsequently, the polymerase pries the double strand apart and starts synthesizing a new strand of RNA which pairs complementarily with one of the strands on the DNA (the *template strand*). The new RNA sequence is thus identical to the other DNA strand (the *coding strand*). The RNA is produced in the direction $5'$–$3'$, implying that the template

strand is read in the direction $3'$–$5'$ during transcription. Once the first few nucleotides of the RNA have been synthesized, the polymerase disassociates from the transcription factor proteins, and the polymerase starts moving along the gene body, transcribing it as it goes (this may require the presence of other transcription factors called *activators*, which are recruited by *enhancer* motifs elsewhere on the DNA).

Eukaryotic chromosomes are very long; human chromosome 1 is around 8.5cm stretched from end to end and, to fit into the cell, is tightly packed into a space-efficient conformation. To achieve this, DNA is coiled around *histones*, small protein complexes, to form *nucleosomes*. Too tight packing, however, has the side-effect of making the DNA inaccessible to the transcription machinery. It is thus a common feature of gene regulation to control the chromatin structure, and thus to control the accessibility of the DNA for transcription factors and the polymerases. In addition to enhancers and promoters, chromatin structure and the modification of histones thus regulate the activity of genes.

Finally, the mRNA produced by the transciption from DNA is used by *ribosomes* to produce a amino acid chain in a process that is called *translation*. This chain later folds into an active protein to perform a biochemical function in a cell.

2.5 The genetic code

The process by which proteins are created from mRNA transcripts is more complex than the one-to-one transcription of DNA into RNA, which after all use a common alphabet to encode the information they carry. By contrast, the *translation* of mRNA transcripts into proteins requires a *code* to interpret the genetic information.

There are 20 different amino acids that are encoded by just 4 different nucleotides. To allow this, several nucleotides must be combined to form a larger unit coding for an amino acid. In the universal *genetic code*, shared by all known species, this is accomplished by grouping three consecutive nucleotides together to form non-overlapping, ungapped *triplet codons* along the mRNA. This results in $4^3 = 64$ possible codons, more than three times the number of

amino acids. As a consequence, the genetic code is *degenerate*: most amino acids can be encoded by more than a single codon.

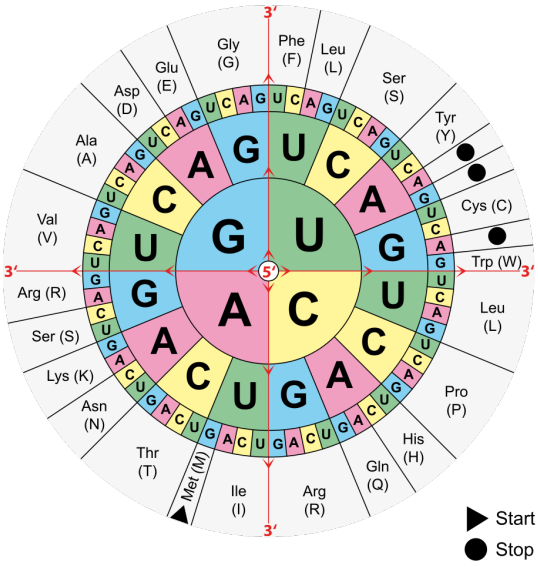


Figure 2.2: A schematic representation of the genetic code where the RNA notation is used. Image by Mouagip [Public domain], via Wikimedia Commons.

Codons furthermore serve as control points by defining where the translated sequence on the mRNA starts and ends. The codon `AUG`, in addition to encoding the amino acid methionine, also marks the start of the coding sequence. Three codons do not encode any amino acid, and instead signal the end of translation (`UAA`, `UAG`, `UGA`). As a consequence, every coding sequence starts with `AUG`, ends with one of the stop codons, and has a length divisible by 3. Figure 2.2 contains a representation of the genetic code, which is valid, with only minor variations, for all domains of life.

2.6 Human Genome Variation Society Nomenclature

Since the 1990s [Beaudet and Tsui, 1993, Beutler, 1993] attemps have been made to capture the description of genomic variation. These discussions culminated in the Human Genome Variation Society (HGVS) Nomenclature [den Dunnen et al., 2000]. Here a set of rules is presented to describe a restricted set of genomic variations. Many additions and revisions have since been made to cater for more complex variation, resulting in the current version (15.11) of the HGVS Nomenclature [den Dunnen et al., 2016] (`http://varnomen.hgvs.org`). These recommendations have world-wide acceptance as the standard nomenclature for clinical diagnostics and are also widely used in other fields.

The principal characteristics of the nomenclature aim for stability, meaningfulness, memorability and unambiguity. The nomenclature is documented using natural language and is mainly example-driven (see `http://varnomen.hgvs.org`). A formal definition of its syntax has been constructed in [Laros et al., 2011], however a more formal definition of its semantics is missing. In this dissertation we frequently refer to the HGVS Nomenclature and when doing so we have a clear subset of its rules in mind. Usually, we will restrict ourselves to so-called genomic descriptions, i.e., descriptions based upon a genomic sequence, e.g. a chromosome, without any additional annotation for coding regions or genes. Many variants are commonly described on the gene level, i.e., including annotation for coding regions, exons and genes. The Mutalyzer tool suite [Wildeman et al., 2008] provides a way of converting genomic descriptions to other positioning schemes. Furthermore, we consider only rules that will result in a *proper description* in the sense that given a reference sequence and a description one should be able the construct the observed sequence. This excludes some of the constructions that are part of the HGVS nomenclature. Most prominently descriptions dealing with ranges of positions and uncertainty. To give more feeling for what proper descriptions are it helps to notice that in most of the cases in this dissertation we use an imaginary substitution operator which substitutes a certain substring in the reference sequence into a given string. Note that the substitution operator in the HGVS nomenclature deals only with single nucleotide substitutions (a

special case of the imaginary substitution operator). Indeed, most types of variants within the HGVS nomenclature are special cases of this imaginary operator. For descriptions on the DNA level we also need an operator that gives us the *reverse complement* of a string. In Table 2.1 we give some examples of the HGVS variant descriptions.

Table 2.1: Examples of typical HGVS variant descriptions.

Substitution	`g.1234A>C`	A single nucleotide substitution on a given position in the reference sequence.
Deletion	`g.1234_2143`	A deletion of one or more nucleotides from the reference sequence.
Inversion	`g.1234_2143inv`	The reverse complement of the reference sequence.
Insertion	`g.1234_1235ATTTA`	An insertion of a sequence in the reference sequence.
Duplication	`g.1234_1243dup`	One or more nucleotides are inserted directly $3'$ of the original copy of that sequence in the reference sequence.
Deletion insertion	`g1234_2143delinsA`	One or more nucleotides are replaced by an inserted sequence.

In the most of the chapters of this dissertation we make extensive use of the HGVS nomenclature and due to the nature of the topics covered in these chapters we frequently propose small modifications and additions to the nomenclature. Most of these are, as now, not officially part of the HGVS nomenclature.

Chapter 3

An Efficient Algorithm for the Extraction of HGVS Variant Descriptions from Sequences

Unambiguous sequence variant descriptions are important in reporting the outcome of clinical diagnostic DNA tests. The standard nomenclature of the Human Genome Variation Society (HGVS) describes the observed variant sequence relative to a given reference sequence. We propose an efficient algorithm for the extraction of HGVS descriptions from two sequences with three main requirements in mind: minimizing the length of the resulting descriptions, minimizing the computation time, and keeping the unambiguous descriptions biologically meaningful.

Our algorithm is able to compute the HGVS descriptions of complete chromosomes or other large DNA strings in a reasonable amount of computation time and its resulting descriptions are relatively small. Additional applications include updating of gene variant database contents and reference sequence liftovers.

The algorithm is accessible as an experimental service in the Mutalyzer program suite (`https://mutalyzer.nl`). The C++ source code and Python interface are accessible at: `https://github.com/mutalyzer/description-extractor`.

3.1 Introduction

The Human Genome Variation Society publishes nomenclature guidelines [den Dunnen et al., 2000] for unambiguous sequence variant descriptions in clinical reports, the literature and genetic databases. The Mutalyzer program suite [Wildeman et al., 2008] has been built to automatically check and correct these variant descriptions. As many complex variants are supported, the corresponding descriptions are not always straightforward to construct, justifying the need for the automatic extraction of HGVS descriptions by comparison of the sequence observed in an individual to the reference sequence specified in guidelines and databases. Here we approach this from an informatics perspective as a string comparison problem.

Consider two DNA strings: R, the reference string and S, the sample or observed string:

$$R = \texttt{ATGAT GATCAGATACAGTGTGATACAGGTAGTTAG ACAA}$$
$$S = \texttt{ATGATTTGATCAGATACA TGTGATACCGGTAGTTAGGACAA}$$

The string S can be rewritten in terms of string R by using the HGVS description:

`g.[5_6insTT;17del;26A>C;35dup]`

The *string-to-string correction problem* calculates the distance between two strings as measured by the minimum cost of a sequence of *edit operations* needed to transform one string into the other. The traditionally allowed edit operations [Wagner and Fischer, 1974] are exchanging one symbol of a string for another: a *substitution* indicated using > between symbols (`26A>C`), deleting a single symbol from a string: a *deletion* indicated using abbreviation `del` (`17del`), and inserting one symbol: an *insertion* using abbreviation `ins` (`5_6insTT`). There is a specific case: insertion of previous symbol(s) is described with HGVS term *duplication* using abbreviation `dup` (`35dup`). The string-to-string correction problem has been extended on in numerous occasions [Wagner and Lowrance, 1975, Tichy, 1984] usually allowing more powerful edit operations. Here, we solve another extension of this problem by defining additional edit operators especially suited to the HGVS nomenclature.

Formally, our extension can be defined as follows. Given two strings R and S over the finite alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, and a set of edit operators with their corresponding (non-negative) weights, calculate a sequence of edit operations that transforms a reference string R into a sample string S with a minimum cost with regard to the weights of the operations given in Table 3.1. The weights in Table 3.1 are based on the textual length of the HGVS nomenclature. Note that the length of the description of the position is dependent on the position, i.e., towards the end it takes more symbols to describe the position, therefore we will parameterize all weights making them independent of the positions.

Traditionally, the edit operations are defined on single symbols. To provide a more intuitive way of describing variants, we extend these operations in a natural way allowing use of substrings rather than individual symbols. Note that, in contrast to the insertion operator, the deletion operator on multiple symbols is not dependent on the length of the deleted substring, thereby creating an asymmetry between insertion and deletion.

In addition to the traditional edit operators we define two additional operators: *inversion* (HGVS abbreviation: `inv`) matches the *reverse complement* of the string and *transpositions*.

3.1.1 Transpositions

Here we define transpositions to be substrings which are copies of substrings found either elsewhere in the matched string or elsewhere in the same string. As we are interested in calculating concise descriptions, we will only consider insertions to be candidates for transpositions. This will produce favorable results especially in the case of long insertions that can be described as long transpositions as their weights are independent of the length of the inserted substrings. Furthermore, we allow some variants within these transpositions yielding *composite* transpositions, e.g.:

```
g.[5_6ins[GG;17_45;inv46_78]]
```

This composite transposition consists of three parts: a regular insertion of `GG`, a transposition of a substring of the reference sequence from position 17 to 45

Table 3.1: Edit operators for HGVS descriptions with their corresponding weights.

Operator	HGVS Description	Weight
Deletion (single)	p`del`	$x + 3$
Deletion (multiple)	$p_{\text{start}}\_p_{\text{end}}$`del`	$2x + 4$
Deletion/insertion (single)	p`delins`w	$x + 6 + \lvert w\rvert^{\dagger}$
Deletion/insertion (multiple)	$p_{\text{start}}\_p_{\text{end}}$`delins`$w$	$2x + 7 + \lvert w\rvert^{\ddagger}$
Insertion	$p_{\text{start}}\_p_{\text{end}}$`ins`$w$	$2x + 4 + \lvert w\rvert^{\ddagger}$
Inversion	$p_{\text{start}}\_p_{\text{end}}$`inv`	$2x + 4$
Substitution	pc_1`>`c_2	$x + 3$
Transposition	$p_{\text{start}}\_p_{\text{end}}$`ins`$[p_{\text{start}}\_p_{\text{end}}]$	$4x + 4$
Inverse transposition	$p_{\text{start}}\_p_{\text{end}}$`ins`$[p_{\text{start}}\_p_{\text{end}}$`inv`$]$	$4x + 7$

where x is the weight of a position description independent of the actual position.

† $w \in \Sigma^*$, with $|w| > 1$

‡ $w \in \Sigma^*$, with $|w| > 0$

followed by a transposition found on the reverse complement of the reference sequence, i.e., an inverse transposition. Note that the alternative would require the insertion of 62 nucleotides.

The remainder of this chapter is organized as follows. In Section 3.2 we introduce an algorithm to efficiently compute the HGVS description between two strings. Section 3.3 describes the experiments, followed by a discussion of the results in Section 3.4 and the conclusions in Section 3.5.

3.2 Methods

In order to automatically construct HGVS descriptions we propose an extraction algorithm. The three main requirements considered for this algorithm are:

1. The length of the descriptions — we try to minimize these;

2. The computational speed — in order to be practically useful we consider a maximum total computation time of 1 hour for chromosome 1 of the human genome on a desktop PC (3.4 GHz and 16 GB RAM). Although this specific timing criterion is arbitrary it serves as a indication for a responsive desktop environment;

3. The (biological) meaning of the descriptions — given that this algorithm is developed for genetic data, we want the descriptions to be as close as possible to the intuition of the people using them.

3.2.1 Extraction algorithm

A trivial way to describe the sample string in terms of the edit operations from the reference string, is to give the substitution of the whole reference string with the sample string by means of the deletion/insertion operator. This gives us an upper bound on the length of the description. We can stop recursively cutting the strings at the moment when the resulting description exceeds the trivial description or when we can decide that every possible description from this point on will result in a longer description.

The underlying idea of the extraction algorithm is to divide the string to be described into a sequence of unaltered regions and altered regions. The altered regions are then described according to the HGVS nomenclature. In order to minimize the length of the resulting descriptions, we apply a *greedy* approach by choosing the longest possible unaltered regions. Note that this is a heuristic which implies that it might be possible to find a more concise description by choosing a smaller unaltered region.

The algorithm is formulated recursively: given two strings R and S find the longest string that is a substring of R as well as S. Remove this string from

the problem, and continue recursively with both prefixes R_{pre} and S_{pre} and both suffixes R_{suf} and S_{suf}. The recursion ends when either of the two strings is empty or no common substring could be found, see Figure 3.1. In case of an empty reference string and a non-empty sample string, the corresponding variant is an insertion. When the sample string is empty and the reference string is not, the corresponding variant is a deletion. If no common substring could be found, depending on the length of both strings we deal with a substitution in case of a single nucleotide or a larger deletion/insertion.

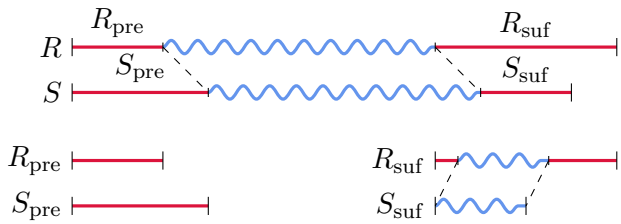


Figure 3.1: Graphical representation of the extraction algorithm with reference string R and sample string S, with the recursion showing a common substring in the suffixes (suf), but not in the prefixes (pre). The wavy lines denote the LCS during that iteration.

3.2.2 Finding the Longest Common Substring

In this section we explain the traditional approach for finding the longest common substring between two strings as an introduction to the more efficient version we present in Section 3.2.3.

The problem of finding the *longest common substring(s)* (LCS) between two (or more) strings is a well studied problem [Gusfield, 1997]. Traditionally, a dynamic programming approach for finding the LCS is used. Based on the recurrence relation (3.1), a table M is built containing at each position (i, j) the length of the longest common suffix between both prefixes.

Equation (3.2) is used to find the length of the longest common substring.

Together with the position (i, j) we can easily find the actual string.

$$M(S_{1..i}, R_{1..j}) = \begin{cases} M(S_{1..i-1}, R_{1..j-1}) + 1 & \text{if } S_i = R_j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

$$\text{LCS}(S, R) = \max_{1 \le i \le |S|, 1 \le j \le |R|} M(S_{1..i}, R_{1..j}) \tag{3.2}$$

In order to illustrate the mechanisms of finding the LCS, we will present an example. Let $R =$ `AACACTTA`, and $S =$ `ACTAACACTT`. We construct M according to the recurrence relation (3.1) as shown in Table 3.2. We fill M from top to bottom, and from left to right. If the symbols on position (i, j) match, we look at position $(i - 1, j - 1)$ and extend the matched suffix. For instance, position $(3, 6)$ has 3, because position $(2, 5)$ has 2 and `T` matches `T`.

Table 3.2: Dynamic programming approach for finding the longest common substring. Here, the LCS is `AACACTT`, with length 7.

M	A	A	C	A	C	T	T	A
A	1	1	0	1	0	0	0	1
C	0	0	2	0	2	0	0	0
T	0	0	0	0	0	3	1	0
A	1	1	0	1	0	0	0	2
A	1	2	0	1	0	0	0	1
C	0	0	3	0	2	0	0	0
A	1	1	0	4	0	0	0	1
C	0	0	2	0	5	0	0	0
T	0	0	0	0	0	6	1	0
T	0	0	0	0	0	1	7	0

The number of rows in M corresponds to the length of S, while the number of columns corresponds to the length of R. By filling M we deduce the runtime and memory complexity of this algorithm: $O(|R| \cdot |S|)$. Usually $|R| \approx |S|$, giving a quadratic time behavior for this algorithm. We can easily reduce the required

amount of memory by storing only the current and previous row of table M, which gives us a memory bound of $O(\min(|R|, |S|))$.

Although this dynamic programming approach seems similar to the Smith-Waterman algorithm [Smith and Waterman, 1981] for local alignment, it is significantly different. In this phase of the extraction algorithm we focus only on finding the LCS. This permits us to use more powerful and non-local edit operators, i.e., inversions and transpositions which are not possible within the local alignment algorithm.

3.2.3 Finding the LCS more efficiently

In theory an instance of *generalized suffix trees* could be exploited giving us a linear bound on runtime. However, the implementations are impractical both in memory requirements as well as having large constants in the linear runtime. Instead, we will present an alternative LCS retrieval method based on the traditional dynamic programming approach in Section 3.2.2.

Although for application to chromosomal sequences we have to calculate the LCS of two large strings, we expect that these strings within one species would be very similar to each other. We expect the LCS of those strings to be very large compared to the length of the strings. Using this knowledge we propose to encode the strings into a higher alphabet. We split both string into substrings of length k, called k-mers, one string into non-overlapping k-mers and the other into overlapping k-mers. Using a k-mer representation is a well-known optimization for sequence alignment [Compeau et al., 2011].

The size of the table required is greatly reduced by the use of non-overlapping k-mers. It is, however, impossible to split both strings into non-overlapping k-mers, because it would impose a constraint on the starting position of the LCS to be found: only a LCS starting on a kth position can be found. By splitting one string into overlapping k-mers we remove this constraint while still reducing the table size.

In Table 3.3 we show the tables M_2 and M_3 constructed for the same example as given in Table 3.2 by using a modified version of the recurrence

relation (3.1):

$$M_k(S_{1..i}, R_{1..j}) = \begin{cases} M_k(S_{1..i-k}, R_{1..j-1}) + 1 & \text{if } S_i = R_j \\ 0 & \text{otherwise} \end{cases}, \tag{3.3}$$

where S_i and R_j are k-mers.

In order to calculate the value of position $(7, 2)$, we have to look at the position $(7 - 3, 2 - 1)$ to extend the k-mers matched so far. Equation (3.2), adapted in the natural way, can be used to extract the LCS based on k-mers. In this case it yields the LCS `AACACT` with length 6. Consequently, we have to extend the found LCS, possibly at both ends to find the actual LCS of length 7 as a post-processing step. In general, the actual LCS can be extended $k-1$ symbols to the left, and $k-1$ symbols to the right. This implies that for $k > 1$ the LCS can be found at a position in the M_k table with a sub-optimal value. To be precise: one less than the maximum value found using Equation (3.2). All these positions have to be considered for the LCS as well.

Table 3.3: Dynamic programming approach with overlapping and non-overlapping 2-mers and 3-mers.

M_2	AA	CA	CT	TA
AC	0	0	0	0
CT	0	0	1	0
TA	0	0	0	1
AA	1	0	0	0
AC	0	0	0	0
CA	0	2	0	0
AC	0	0	0	0
CT	0	0	3	0
TT	0	0	0	0

M_3	AAC	ACT
ACT	0	1
CTA	0	0
TAA	0	0
AAC	1	0
ACA	0	0
CAC	0	0
ACT	0	2
CTT	0	0

In comparison to the original table M, the M_k table is much smaller: $(|S| - k + 1) \times \lfloor |R|/k \rfloor$. If we can swap the roles of R and S freely, it is

advantageous to choose R to be the longest string. Again, we only have to store a part of this table: the current row and the k previous ones. All of these rows contain fewer elements than those in table M. The memory constraints remain approximately the same as for the original algorithm. The runtime, however, is greatly reduced for large values of k.

3.2.4 Choosing the size of the of k-mers

If we compare Table 3.2 and Table 3.3, it appears that we cannot find all arbitrary common substrings of R and S. For instance, the substring `ACT` starting in R at position 9 and in S at position 7 is not present in Table 3.3 due to an unfortunate misalignment in the non-overlapping k-mers. Moreover, all common substrings with a length less than k are not present at all. To be certain to find a common substring of length ℓ, k has to be at most $\lceil \ell/2 \rceil$. Therefore, we can consider k to be a guess for the expected length of the LCS between R and S.

To achieve the best performance of this algorithm, the initial value for k has to be chosen carefully. On one hand we like k to be as large as possible to reduce the runtime as much as possible. On the other hand k has to be small enough compared to the LCS between the two strings. In general we will not know the exact length of the LCS.

In case the algorithm returns no result, we lack the guarantee of the traditional approach, that there is no common substring between both strings. If k is chosen too large, the whole table will contain zeroes or ones and the algorithm fails to produce a result. To find the LCS, we have to reduce the value of k and run the algorithm again until a result is returned or the value of k falls below a certain threshold. In general, this threshold can be 1 which guarantees that there exists no common substring between both strings. However, this is impractical for large strings. Usually, the threshold can be set at the expected length of the LCS between two random strings over alphabet Σ, trivially bound by $2 \log_{|\Sigma|} n$ for strings of length n [Abbasi, 1997].

3.2.5 Adapting the extraction algorithm for inversions, transpositions and inverse transpositions

So far the extraction algorithm in Figure 3.1 handles only variants of the deletion, insertion, and substitution operations. To add support for the inversion operator, we have to run the LCS algorithm twice. First, the sample string is matched to the original reference string. Second, it is matched against the reverse complement of the reference string (in every instance of the recursion). If the LCS is found on the reverse complement, the algorithm picks this LCS and removes it from the solution. In the exceptional case of a tie between the length of a regular LCS and the length of a reverse complement LCS, the algorithm prefers the regular one, because of the higher weight associated with a reverse complement match. In the next step of the recursion a new decision will be made on whether to use the original or a reverse complement LCS independent of the current choice. Note that the complexity of the algorithm does not change essentially as we do twice the amount of work.

In order to find useful transpositions, we consider all insertions of a certain length. In practice, insertions of two base pairs will usually not be considered to be transpositions as all occurrences of two base pairs will be present elsewhere. With increasing length of the insertions the probability that these exact sequences are found elsewhere diminishes quickly. Therefore, if we are able to locate these sequences elsewhere, we can be confident that they are indeed transpositions. Instead of looking for the exact sequences, we use a modified recursive instance of the extraction algorithm to find transpositions with small variations. The main difference between the regular extraction algorithm and the modified algorithm proposed here is that deletions within a transposition are not meaningful, i.e., we just describe the actual insertions either as regions to be found elsewhere in the string or as regular insertions. Likewise, inverse transpositions are found by matching against the reverse complement string.

3.3 Experiments

We performed computer experiments to demonstrate the performance of our proposed algorithm both in terms of speed and the quality of its output. In the first experiment we will focus on the performance of the extraction algorithm on large DNA strings, i.e., whole human chromosomes. The second experiment aims to minimize the resulting descriptions in a real-life case study. The final experiment shows the biological quality of the resulting descriptions.

In all experiments we used a fixed initialization and reduction scheme for k when the algorithm fails to return a solution, as explained in Section 3.2.4. We initialize k to $|R|/4$; in case of no solution we reduce $k \leftarrow k/3$. This seems to be a good balance for maximizing k and miniziming the amount of re-runs for the LCS_k algorithm. On average 1 to 2 re-runs are sufficient.

For the transposition cut-off discussed in Section 3.2.5, we specify a threshold of 10% of the length of the inserted string. Any matched regions smaller than this length are considered to be uninteresting as transpositions and are described as regular deletions/insertions. Modifying this cut-off will greatly affect the runtime of the algorithm. Again, for our experiments, this cut-off strikes a good balance between runtime, description length, and biologically interesting patterns.

3.3.1 Performance on large DNA strings

To demonstrate the usefulness and speed of our proposed algorithm we used all chromosomal RefSeq sequences from human genome build NCBI36 (GCF_000001405.12), GRCh37 (GCF_000001405.13) and
GRCh38 (GCA_000001405.15) and performed three extraction experiments:

1. NCBI36 (sample) vs. GRCh37 (reference);
2. NCBI36 (sample) vs. GRCh38 (reference);
3. GRCh37 (sample) vs. GRCh38 (reference).

We extracted the HGVS descriptions of the differences of the respective sample sequences relative to the respective reference sequences per chromosome with

a total computation time of about 40 hours, see Figure 3.2.

As a preprocessing step we replaced all sequential occurrences of `N` with a single `N`. Large sequences of `N` are commonly found at the starts and ends of chromosomes (telomeres) and at their centers (centromeres). We particularly wanted to avoid transposition matching of sequences of `N` as they yield no information.

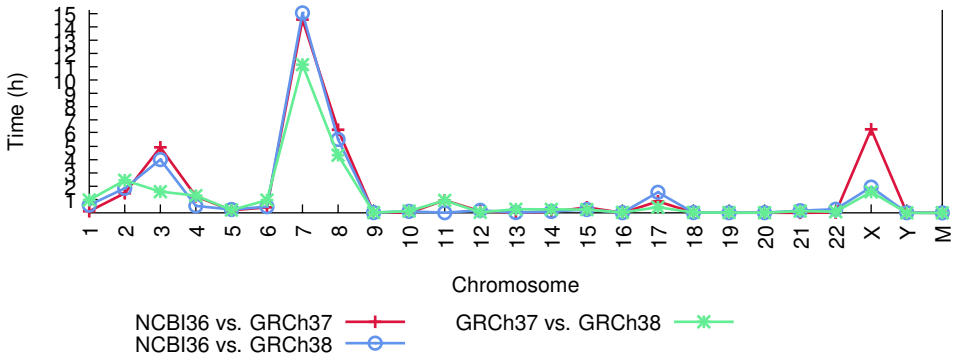


Figure 3.2: Performance of the extraction algorithm per chromosome on a desktop PC (3.4 GHz and 16 GB RAM).

In Figure 3.3 we observe that the maximum description length for any chromosome is about 1 MB. The descriptions can be calculated in at most 1 hour for most chromosomes except for chromosomes 5, 7, 8, and X. Here, we observe a large number of relatively small insertions which are just large enough to be considered for the transposition extraction. This process is very expensive in terms of calculation time, as a whole chromosome needs to be matched against a small string, eliminating the speed-up gained when using a large k.

There seems to be no obvious relation between the calculation time and the resulting description length; a longer calculation time does not always result in a more concise description. Again, small insertions seem to contribute most to this phenomenon. Often the expensive transposition extraction process is started, but the resulting description in terms of transpositions is, in the end, longer than the trivial description. This results in an increase in computation

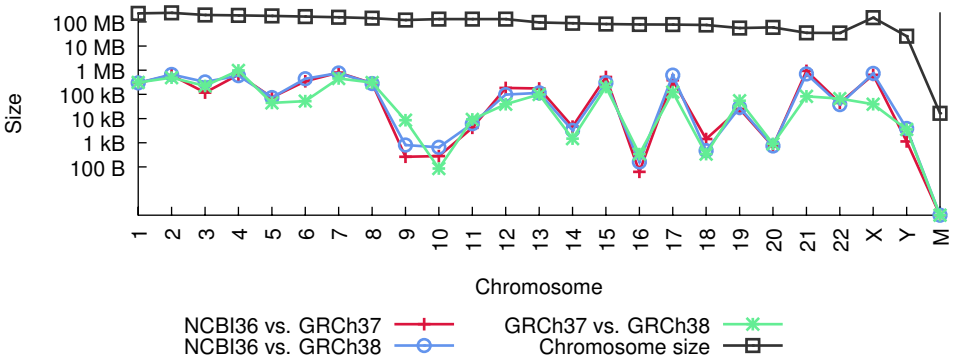


Figure 3.3: HGVS Description lengths of the extraction algorithm per chromosome.

time as well as in the description length.

We should mention that the case of description of one genome build relative to an other is a very artificial example. In each new version of the human genome information is added, i.e., gaps representing unsequenced regions have been replaced with regions that had not been sequenced before and assembly errors have been corrected. This results in multiple large insertions. Also, these descriptions yield no useful biological knowledge. However, we can give an estimate of the amount of information added with every new build.

Parallelization of the algorithm is trivially possible by using threads for each recursive call. The task of efficiently partioning the work over a fixed number of threads is non-trivial. The current recursive definition implies that many calls will terminate relatively quickly. The overhead of starting threads in these circumstances should be considered carefully. Our algorithm in its current form is ill-suited for distribution over multiple machines. Apart from the design of our algorithm we also have to provide an efficient way of accessing global data as the algorithm uses non-local operators. Moreover, one of our primary design criteria is the ability of efficiently generating variant descriptions within a desktop environment.

In Figure 3.4 we present the distribution of the HGVS operators from the description of Chromosome 2 (NCBI36 vs. GRCh37). This distribution is

representative for the distribution of the operators for most of the chromosomes in this experiment (note that Chromosome M has no variants).

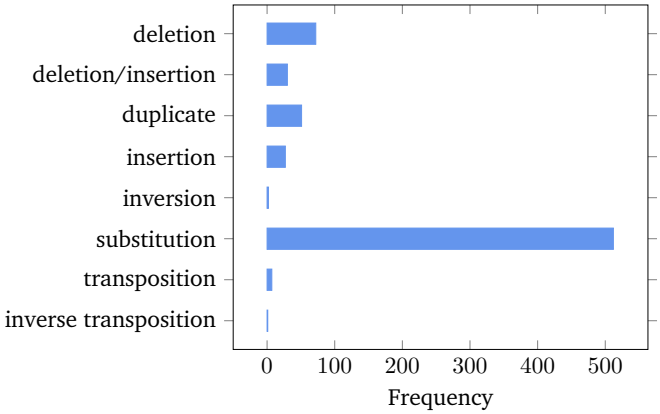


Figure 3.4: The distribution of HGVS operators for Chromosome 2 (NCBI36 vs. GRCh37).

The distribution in Figure 3.4 shows that almost 74% of all variants are substitutions. The insertion operators contribute most to the length of the descriptions (data not shown). The individual variants of composite transpositions are partitioned into their respective operators, e.g., the transposition `12_13ins[17_51;GC;50_99;CTCTG]` contains two transpositions, and two insertions.

3.3.2 Automated description extraction using sequences from a gene database

For this experiment we used the IMGT/HLA Database [Robinson et al., 2014] from which we extracted the sequence of $3{,}588$ HLA-B variant genes. For most of these sequences allele descriptions in HGVS-like format are provided using coding DNA numbering with RefSeq Gene reference sequence NG_023187.1 (see for example: `https://ebi.ac.uk/cgi-bin/ipd/imgt/hla/get_allele_hgvs.cgi?B*73:01`). As all sequences are between 500 and $1{,}000$ bp long, calculation time is not an issue and is in fact dominated by disk access times. For

this experiment it took about 1 hour to automatically generate all HGVS descriptions.

The original descriptions contain predominantly substitutions. For substitutions that are very close together it is often more concise to describe them using a deletion/insertion operator. The HGVS nomenclature forbids the occurrence of two adjacent substitutions. However, these are commonly found in the original descriptions. The original descriptions never have deletions at the beginning or end of the sequence while these variants are all captured by the automatic extraction process. Because of the missing deletions the resulting description length of the automated extracted description can be longer than its corresponding original one. If we disregard these deletions, the automatically derived description is either the same or of (much) smaller length. Finally, we have observed some irregularities in the original descriptions with regard to the HGVS nomenclature, e.g., `[960_961dupT;]` which contains two mistakes and an inaccuracy: (1) only one nucleotide is duplicated, so there is no need for a range of positions, (2) the nucleotide letter(s) do not have to be present for duplicates, and (3) as there is no variant following the duplicate no separating symbol (;) is needed. We have communicated the results of our description extraction with the curators of the IMGT/HLA Database.

3.3.3 Replacing reference sequences for gene variant databases

Gene variant database curators need to update gene-centered reference sequences (predominantly RefSeq Gene files) when new (improved) versions are generated following the release of a new genome build. The current algorithm can help: variant sequences can be generated from the original descriptions using the Mutalyzer Name Checker. These sequences can be compared with the new RefSeq Gene sequence leading to updated HGVS variant descriptions. These descriptions can replace the old ones in the database.

3.4 Discussion

In this section we introduce two additional qualities of automatically generated HGVS descriptions especially when used in genomic databases.

3.4.1 Compression

HGVS descriptions can be an attractive alternative for compressing DNA sequences, especially in a database containing sequences with high similarity that can be described using a single reference sequence. Often the standard reference genome can be used. All instances in the database can be stored by using their HGVS descriptions instead of their sequences and (optionally) one copy of the reference sequence. The difference between the original chromosome size and its corresponding description length is large: up to a million times smaller, see Figure 3.2. To give an impression of the overall compression efficiency, storing the complete human genome requires approximately 3 GB, while storing only the descriptions will take approximately 6 MB per instance giving a reduction 466 times. [Brandon et al., 2009] introduced a similar way of compressing genomic sequence data. They achieved similar results in terms of the compresssion factor as our method. As they focus on developing a compression algorithm, they used a binary encoding scheme for frequent partial variants. In this respect their algorithm differs from ours: we focus on the actual variants and we describe the complete variants in a human readable form.

Traditional compression techniques such as gzip will reduce the size of the human genome to approximately 800 MB. Apart from a much better compression rate, the HGVS format is human readable. Furthermore, many useful queries, e.g., determining the presence of a substitution, can be performed directly on the HGVS descriptions without the need for decompression.

3.4.2 Transitivity

In principle we could also transform descriptions generated using one specific reference string to descriptions versus other reference strings. This is a

potentially powerful operation for large genomic databases. It enables the conversion of entire databases to a new version of the reference genome in considerably less time than generating descriptions versus this new reference genome from scratch.

This transformation can be done by generating the HGVS description of the original reference string versus the new reference string, and then computing the new HGVS description for each instance by combining its description with the description of the reference genomes. This results in a linear (in terms of the description) amount of work for each instance. The actual implementation of the merging is beyond the scope of this chapter. However, to give an intuition for a possible implementation we provide a small example. Consider the HGVS descriptions `g.[5_14inv]` and `g.[3T>C;9G>C]`. The merging of non-overlapping variants is trivial. Positions might have to be offset based on the length of insertions and deletions in the prefix. For overlapping variants we can either construct the corresponding trivial deletion/insertion, i.e., `g.[5_14delinsCGACCGAT]` or alternatively split the inversion into two inversions separated by a substitution: `g.[10_14inv; 9G>C;5_8inv]`. Although the resulting description is a valid HGVS description, a more concise description might be found when running the extraction algorithm directly.

3.5 Conclusion

We introduced an algorithm to extract HGVS descriptions from raw DNA sequences with respect to reference sequences. We made this algorithm computationally efficient for highly similar strings by introducing an alternative version of the classic LCS algorithm using overlapping and non-overlapping k-mers. We showed that the combination of these algorithms is able to compute the HGVS descriptions of large DNA strings in a reasonable amount of computation time and that the resulting descriptions are relatively small. The HGVS descriptions yielded by the extraction algorithm are fully compliant with the Mutalyzer tool suite. The Name Checker tool can be used to generate the original sample string from the description.

We proposed to extend the HGVS nomenclature with the transposition

operator as it can greatly reduce the lengths of descriptions, while still being able to efficiently compute these transpositions.

In addition to having a canonical algorithm for generating HGVS descriptions we have shown that these descriptions are useful in genomic databases for their compression and transitivity properties. The automatic extraction of descriptions will be of great value to curators of existing databases: it makes updates using new versions of reference sequences or of the nomenclature and correction of HGVS descriptions very easy.

3.5.1 Future work

Nesting of variants has been proposed in an extension of the HGVS nomenclature [Taschner and den Dunnen, 2011]. Our extraction algorithm does not support nesting (with the exception of complex transpositions). A possible extension of the extraction algorithm could be made towards finding simple nested variants.

Breakpoint sequences observed with NGS sequencing technology also need to be described in sufficient detail to allow reconstruction of their sequence. The HGVS nomenclature committee is working on new guidelines involving junctions of more than one chromosome. The current version of our algorithm does not yet support transpositions involving more than one chromosome.

Other types of strings can be considered as well. We are mainly focussing on an extraction algorithm for amino acid sequences to describe changes in proteins using an altered set of edit operators.

Chapter 4

Extraction of HGVS Variant Descriptions from Protein Sequences

Frameshift variants are an important class of variants when considering protein sequences. Small deletions/insertions in DNA sequences may result in (large) frameshifts thereby large changes in the protein sequence and its function. We propose a method of finding and annotating frameshift variants, in the spirit of the standard nomenclature of the Human Genome Variation Society (HGVS), from two protein sequences without considering the underlying DNA sequences. Our method is able to efficiently compute HGVS descriptions for protein sequences with frameshift annotations for all species codon table. Furthermore, we show that this method can be effectively used to find promising novel evolutional events. We propose an addition to the HGVS nomenclature for accommodating the (complex) frameshift variants that can be described with our method. Our method is available in the Description Extractor package for Mutalyzer: `https://pypi.python.org/pypi/description-extractor`. The C++ source code and Python interface are accessible at: `https://github.com/mutalyzer/description-extractor`.

4.1 Introduction

The Human Genome Variation Society publishes nomenclature guidelines [den Dunnen et al., 2000] for unambiguous sequence variant descriptions used in clinical reports, literature and genetic databases. To check and interpret these descriptions the Mutalyzer program suite [Wildeman et al., 2008] has been built with as main purpose the automatic checking disambiguation and correction of variant descriptions. In [Vis et al., 2015] an efficient algorithm for automated DNA/RNA variant description extraction is presented. Here, we propose an extension of this algorithm which provides the automated description extraction of protein sequences without direct knowledge of the underlying DNA/RNA sequences. In particular, it provides *frameshift variant* detection. A frameshift variant is introduced by the insertion or deletion in a coding sequence which length is not divisible by three. In addition, we can annotate protein sequences formed by an inversion and shifted variants of inversions as well.

A frameshift variant is a genetic mutation due to insertions or deletions on a DNA sequence that is consequently translated into a protein by encoding each triplet of nucleotides into an amino acid. The key to introducing frameshift lies in this triplet-based structure also known as the *reading frame*. Any insertion of deletion with a length not divisible by three introduces a huge effect on the protein level; all (or most) of the following amino acids will be different from the unmodified ones. In this chapter we do not consider the *transcription* process (DNA to RNA) nor the *splicing* process. The term DNA level means the coding region of the DNA for a particular protein.

As protein descriptions are supported by HGVS, there is a need for their automatic construction. A naive approach could be the HGVS description extractor from [Vis et al., 2015] on the DNA level and convert its output to the protein level. Unfortunately this approach is not feasible. First, many different variants on the DNA level could lead to the same variant on the protein level. There should be a guarantee that all of these DNA level variants are indeed converted to the same protein variant. Given the complexity of these variants it is far from trivial (maybe even impossible) to give such a guarantee. Second,

sometimes the DNA level is not accessible or unknown. In this case, a protein level description would still be useful. Thirdly, considering only the protein level gives us the possibility of allowing for useful annotation in terms of frameshifts.

Usually reading frame modifications are analyzed on the DNA level [Sheetlin et al., 2014], however, we focus here on frameshift detection on the protein level. Our algorithm is closely related to the methods described in [Gîrdea et al., 2010]. There, a graph-based approach is used to perform *back-translation* to the DNA level and induce frameshift mutation from there. Although theoretically our approach is the same, we allow for more types of frameshifts and we simplify the frameshift detection calculation by precalculation of look-up tables such that the frameshift detection can be performed directly on the protein level. Furthermore, the alignment step is not part of the frameshift detection algorithm, but rather performed in advance. In our case the frameshift detection can be seen as a post-processing step on deletion/insertion variants. These optimizations result in an efficient algorithm.

4.1.1 Frameshift variants

We extend the traditional definition of frameshift mutations to cater for more general frameshift variants. Whereas in the traditional definition only two types of frameshift are recognized, i.e., $+1$ and $+2$, we allow also for frameshifts in combination with inversions on the DNA level. Indeed, the reverse complement can be regarded as a frame modification in the protein. Furthermore, the reverse complement can be combined with the usual $+1$ and $+2$ frameshifts bringing the total number of types of frameshifts to five, see Table 4.1.

The $+1$ and $+2$ frameshifts are caused by insertions or deletions with a length modulo $3 \neq 0$. We have more possibilities for the inverse types; the symmetry around the frame boundaries is important. For pure inversions the "overhang", i.e., the number of bases partially covering a codon, of the frame boundaries should be equal. When the overhang is symmetrical the inversed inserted sequence preserves the original frame boundaries. Unequal overhangs result in inverse $+1$ or $+2$ types (see Table 4.1), e.g., if the overhang on the right hand

Table 4.1: The five types of frameshifts with examples on the DNA sequence and the resulting protein sequence. `AB026906.1` is used as a reference sequence.

Type	DNA Variant	Protein Sequence
+1	274_275insC	...MALFPGCSPHSSWSLGPWTSCY*
	274_275del	...MLFPGCSPHSSWSLGPWTSCY*
+2	274_275insCC	...MATIPWLQPSLFMVTGALDKLLL...
	274del	...MTIPWLQPSLFMVTGALDKLLLT...
inverse	274_309inv	...MTMKSEGCSQGIVHWGLGQVVTD...
inverse +1	275_309inv	...MDHEE*
	272_309inv	...NHEE*
inverse +2	276_309inv	...MEP*
	273_309inv	...IP*

side is larger, the inverted sequence is also shifted to the right and vice versa for a left hand side overhang.

4.1.2 Complex frameshift variants

Traditionally when a frameshift is detected the remainder of the protein sequence is not annotated any further. The assumption being that the resulting protein will differ substantially from its reference. However, combinations of variants on the DNA level can alter the frameshift into an other type of frameshift. They can even restore the original reading frame leaving just a part of the protein sequence modified. For instance consider the following combination of DNA level variants: `AB026906.1:c.[274del;288del;301del]`.

With the corresponding protein sequences:

$$R = \texttt{MDYSLAAALTLHGH}$$
$$S = \texttt{MTIPWRSPHF-HGH}$$

The first deletion will induce a $+2$ frameshift variant. This frameshift is later modified by the second deletion into a $+1$ frameshift. The third deletion restores the original reading frame. We propose that the aforementioned example is annotated as:

$$\texttt{2\_7|2;7\_11|1}$$

The remainder of this chapter is organized as follows. In Section 4.2 we formally introduce the problem of frameshift detection and we introduce an algorithm to efficiently compute frameshift variants between two protein sequences. Section 4.3 contains the experiments, followed by a discussion of additional uses and properties in Section 4.4 and the conclusions in Section 4.5.

4.2 Methods

Based on the extraction algorithm introduced in Chapter 3, we added support for frameshift variant detection by performing the extraction algorithm in two phases. The first phase follows the original algorithm: given two strings R and S find the longest string that is a substring of R as well as S. Remove this string from the problem, and continue recursively with both prefixes R_{pre} and S_{pre} and both suffixes R_{suf} and S_{suf}. The recursion ends when either of the two strings is empty or no common substring could be found. For protein sequences we only use the traditional *edit operations*: deletion, insertion and substitution. Transpositions are not considered. Due to the relative short length of protein sequences there is no benefit in using the LCS$_k$ algorithm.

In the second phase, the regions of change (deletions/insertions) are recursively partitioned into frame shifted regions and regions that cannot be described in terms of frameshifts. We adapted the LCS algorithm from finding exact string matches to match all possible (combinations) of frameshifts, again using a *greedy* approach. Note that we can only add the frameshift variants as

annotation; not as a true variant without violating the property of unambiguity. Without the exact amino acid sequence we cannot reconstruct the observed sequence from the reference given its variants.

Formally the problem can be defined as follows. Let $\Sigma_\mathrm{N} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ be the nucleotide alphabet and the amino acid alphabet $\Sigma_\mathrm{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{V}, \mathtt{W}, \mathtt{Y}, *\}$. A *triplet* or *codon* $c = \langle b_1, b_2, b_3 \rangle$, where $b_1, b_2, b_3 \in \Sigma_\mathrm{N}$, is mapped to an amino acid (from Σ_A) by the surjective *translation* function $f \colon \Sigma_\mathrm{N} \times \Sigma_\mathrm{N} \times \Sigma_\mathrm{N} \to \Sigma_\mathrm{A}$. An example of the translation function is given in Table 4.2. In general, the codon table can be different for (applications within) different species as well as within species. For the human genome the codon table in Table 4.2 is commonly used and shall be used as an example further in this chapter.

Table 4.2: An inverse DNA codon table for the human genome.

Amino Acid	Codon(s)	Amino Acid	Codon(s)
A	GCA, GCC, GCG, GCT	N	AAC, AAT
C	TGC, TGT	P	CCA, CCC, CCG, CCT
D	GAC, GAT	Q	CAA, CAG
E	GAA, GAG	R	AGA, AGG, CGA, CGC, CGG, CGT
F	TTC, TTT	S	AGC, AGT, TCA, TCC, TCG, TCT
G	GGA, GGC, GGG, GGT	T	ACA, ACC, ACG, ACT
H	CAC, CAT	V	GTA, GTC, GTG, GTT
I	ATA, ATC, ATT	W	TGG
K	AAA, AAG	Y	TAC, TAT
L	CTA, CTC, CTG, CTC, TTA, TTG	* (stop)	TAA, TAG, TGA
M (start)	ATG		

Now we define the frameshift functions:

$$
\begin{aligned}
fs_{+1}\left(c_1, c_2, c_\mathrm{s}\right) &= \begin{cases} \text{true} & \text{if } \langle b_{1_3}, b_{2_1}, b_{2_2} \rangle = c_\mathrm{s}, \text{with } b_{1_i} \in c_1 \wedge b_{2_j} \in c_2 \\ \text{false} & \text{otherwise} \end{cases} \\
\widehat{fs}_{+1}\left(a_1, a_2, a_\mathrm{s}\right) &= \bigvee_{\substack{\forall c_1 \in f^{-1} a_1 \\ \forall c_2 \in f^{-1} a_2 \\ \forall c_\mathrm{s} \in f^{-1} a_\mathrm{s}}} fs_{+1}\left(c_1, c_2, c_\mathrm{s}\right).
\end{aligned}
$$

The functions for $+2$ frameshifts are analogous to fs_{+1} and $\widehat{fs}_{+1}$.

$$fs_{\text{inv}}(c, c_{\text{s}}) = \begin{cases} \text{true} & \text{if } \langle \text{inv}(b_3), \text{inv}(b_2), \text{inv}(b_1) \rangle = c_{\text{s}}, \text{with } b_i \in c \\ \text{false} & \text{otherwise} \end{cases}$$

where inv is the complement of a nucleotide, i.e., A $\Leftrightarrow$ T and C $\Leftrightarrow$ G.

$$\widehat{fs}_{\text{inv}}(a, a_{\text{s}}) = \bigvee_{\substack{\forall c \in f^{-1} a \\ \forall c_{\text{s}} \in f^{-1} a_{\text{s}}}} fs_{\text{inv}}(c, c_{\text{s}})$$

In Figure 4.1, an example frameshift calculation is given.

$$\begin{aligned} R &= \texttt{MDYSLAAALTLHGH} \\ S &= \texttt{MTIPWRSPHF-HGH} \end{aligned}$$

D	Y		+2	
GAC	TAC	G	ACT	AC
GAC	TAT	G	ACT	AT
GAT	TAC	G	ATT	AC
GAT	TAT	G	ATT	AT

T = {ACA, ACC, ACG, ACT}

$$fs_{+2}(\texttt{GAC}, \texttt{TAC}, \texttt{ACT}) = \text{true} \rightarrow \widehat{fs}_{+2}(\texttt{D}, \texttt{Y}, \texttt{T}) = \text{true}$$

Figure 4.1: Example calculation of a $+2$ frameshift. This example can be continued by calculating $\widehat{fs}_{+2}(\texttt{Y}, \texttt{S}, \texttt{I})$, etc.

The functions for the inverse frameshifts are analogous to fs_{+1} and $\widehat{fs}_{+1}$ with the inverse codon reordering analogous to fs_{inv}.
The results of all frameshift functions can be precomputed and stored in a look-up table. Moreover, all types of frameshift can be simultaneously checked by encoding the five types as a bit array. We use a $|\Sigma_{\text{A}}| \times |\Sigma_{\text{A}}| \times |\Sigma_{\text{A}}|$ look-up table corresponding to the signature of the $\widehat{fs}$ functions. Although the signature

of the inverse frameshift functions is different, we can still use the same look-up table. This renders a frameshift check equivalent to character matching in string comparison. As a consequence frameshift detection is efficient. Note that frameshift types can be overlapping for a certain combination of amino acids, e.g., `P` can be found as either frameshift $+1$ or $+2$ in `SL`.

For each deletion/insertion a frameshift extraction is started based on the aforementioned frameshift functions. Again we follow a *greedy* approach: we select the the longest same-type frameshift. In the event of a multi-type frameshift it is annotated as such. The frameshift extraction is then continued on the remaining prefixes and suffixes of the original deletion/insertions until one of the prefixes of suffixes is empty, much like the original extraction routine. The possibility of having multi-type frameshifts introduces an uncertainty of the actual frameshift type boundary when considering a complex frameshift variant. When we consider a variant where a frameshift $+2$ is altered to a frameshift $+1$ for instance by deleting a single nucleotide on the DNA level we cannot accurately predict the actual frameshift type boundary when some of the amino acids show a multi-type frameshift, i.e., $+2$ and $+1$. The longest part of the complex frameshift will be dominant in this case by including as much amino acids as possible. As we assume no knowledge of the DNA level this is optimal in this situation.

4.2.1 Probability calculation

To add to the frameshift annotation we can estimate the probability of the frameshift by using information from the codon table as well as additional information about the distribution of codon usage in a species. When this additional information is absent, we can approximate the probability by using the distribution of the amino acids and assume a uniform probability distribution for each amino acid over its codons. If this information is also absent, we can still approximate the probability by assuming a uniform distribution of amino acids. This calculation can be refined by incorporating different distributions when comparing two different species. In general, the most simple approximation, i.e., no extra information (only using the codon table), will yield a usable

approximation of the frameshift probability:

$$1 - \frac{1}{|\Sigma_{\mathrm{A}}|^{\ell}}$$

where ℓ is the length of the frameshift. As is apparent, the probability will increase strongly with the length of the frameshift.

Next, we consider an example LR with frameshift $+2$. We take from Table 4.2 $L = \{\mathtt{CTA}, \mathtt{CTC}, \mathtt{CTG}, \mathtt{CTT}, \mathtt{TTA}, \mathtt{TTG}\}$ and $R = \{\mathtt{AGA}, \mathtt{AGG}, \mathtt{CGA}, \mathtt{CGC}, \mathtt{CGG}, \mathtt{CGT}\}$ and we assume a uniform distribution over the codons. The probability calculation follows the original frameshift calculation quite closely. The following frameshifted codons are observed:

$$\begin{aligned}
&8 \times \mathtt{TAC}\,(\mathtt{Y})\\
&8 \times \mathtt{TGC}\,(\mathtt{C})\\
&4 \times \mathtt{TAA}\,(*)\\
&4 \times \mathtt{TCC}\,(\mathtt{S})\\
&4 \times \mathtt{TGA}\,(*)\\
&4 \times \mathtt{TTC}\,(\mathtt{F})\\
&2 \times \mathtt{TCA}\,(\mathtt{S})\\
&2 \times \mathtt{TTA}\,(\mathtt{L})
\end{aligned}$$

This corresponds to the following probabilities $P(c_{\mathrm{s}}|c_1c_2)$:

$$\begin{aligned}
P(\mathtt{C}|\mathtt{LR}) &= \tfrac{8}{64}\\
P(\mathtt{Y}|\mathtt{LR}) &= \tfrac{8}{64}\\
P(*|\mathtt{LR}) &= \tfrac{4+4}{64}\\
P(\mathtt{S}|\mathtt{LR}) &= \tfrac{4+2}{64}\\
P(\mathtt{F}|\mathtt{LR}) &= \tfrac{4}{64}\\
P(\mathtt{L}|\mathtt{LR}) &= \tfrac{2}{64}
\end{aligned}$$

The probability of a frameshifted sequence is the product of the individual probabilities. In the event of a multi-type frameshift the combined probabilities can be precomputed. When we consider the aforementioned example with added codon distributions $P(c)$:

$$\mathtt{L} = \begin{cases} \mathtt{CTA} & 0.22 \\ \mathtt{CTC} & 0.20 \\ \mathtt{CTG} & 0.28 \\ \mathtt{CTT} & 0.15 \\ \mathtt{TTA} & 0.05 \\ \mathtt{TTG} & 0.10 \end{cases} \qquad \mathtt{R} = \begin{cases} \mathtt{AGA} & 0.10 \\ \mathtt{AGG} & 0.05 \\ \mathtt{CGA} & 0.20 \\ \mathtt{CGC} & 0.22 \\ \mathtt{CGG} & 0.15 \\ \mathtt{CGT} & 0.28 \end{cases}$$

We can calculate the frameshift $+2$ probability:

$$P(c_\mathrm{s}|a_1a_2) = \sum_{\forall c_1 \in f^{-1}a_1 \text{ where } b_{1_2}=b_{\mathrm{s}_1}, b_{1_3}=b_{\mathrm{s}_2}} P(c_1) \cdot \sum_{\forall c_2 \in f^{-1}a_2 \text{ where } b_{2_1}=b_{\mathrm{s}_3}} P(c_2)$$

Again we observe the following codons:

$8 \times \mathtt{TAC}\,(\mathtt{Y})$	$(0.22+0.05)(0.2+0.22+0.15+0.28)$	≈ 0.23
$8 \times \mathtt{TGC}\,(\mathtt{C})$	$(0.28+0.1)(0.2+0.22+0.15+0.28)$	≈ 0.32
$4 \times \mathtt{TAA}\,(*)$	$(0.22+0.05)(0.1+0.05)$	≈ 0.04
$4 \times \mathtt{TCC}\,(\mathtt{S})$	$0.2(0.2+0.22+0.15+0.28)$	≈ 0.17
$4 \times \mathtt{TGA}\,(*)$	$(0.28+0.1)(0.1+0.05)$	≈ 0.06
$4 \times \mathtt{TTC}\,(\mathtt{F})$	$(0.05+0.1)(0.2+0.22+0.15+0.28)$	≈ 0.13
$2 \times \mathtt{TCA}\,(\mathtt{S})$	$0.2(0.1+0.05)$	≈ 0.03
$2 \times \mathtt{TTA}\,(\mathtt{L})$	$0.15(0.1+0.05)$	≈ 0.02

With the resulting probabilities on the protein level:

$$P(a_\mathrm{s}|a_1a_2) = \sum_{\forall c_\mathrm{s} \in f^{-1}a_\mathrm{s}} P(c_\mathrm{s}|a_1a_2)$$

Given below:

$P(\mathtt{C}\|\mathtt{LR})$	≈ 0.32
$P(\mathtt{Y}\|\mathtt{LR})$	≈ 0.23
$P(*\|\mathtt{LR})$	$\approx 0.04 + 0.06 \approx 0.11$
$P(\mathtt{S}\|\mathtt{LR})$	$\approx 0.17 + 0.03 \approx 0.20$
$P(\mathtt{F}\|\mathtt{LR})$	≈ 0.13
$P(\mathtt{L}\|\mathtt{LR})$	≈ 0.02

The afore mentioned example is quite extreme in its given (fictitious) codon distribution. Real-life examples are usually much closer to the uniformly distributed estimations.

As before with the frameshift loop-up table, we can precalculate the probability loop-up table as well. This results in an efficient calculation of the frameshift annotations. For implementations using *floating point* arithmetic one should be aware of the small numbers that result from this calculation and its inherently corresponding errors. In our implementation we prefer fixed point calculation using *binary scaling*.

4.2.2 Back-translation

The technique we have employed to calculate frameshifts and its probabilities can also be used for accurately predicting the DNA sequences underlying the protein. The process of translating amino acid sequences to DNA sequences is known as *back-translation*. In general, the surjective codon function makes it difficult to accurately predict the underlying DNA sequence, i.e., there is a lot of ambiguity resulting in an explosion of the possible DNA sequences. Consider the protein sequence `DYSLA`, with no further information we can translate this sequence to its underlying DNA: `GAYTAYWSNYTNGCN`, where we used the degenerate form to encode the nucleotide variation. When we count the number of DNA sequences they amount to $2{,}048$.

When we know (or we just calculated) that the protein sequence `DYSLA` is actually a frameshift $+2$ with the sequence `TIPWR`, we can, for both sequences, construct a much more accurate back-translation: `GACTATTCCYTGGCG`, with only two possible DNA sequences and `ACTATWCCYTGGCGG`, with four DNA sequences. The possibilities for each nucleotide are considerably reduced by the restriction of the partially overlapping codons.

4.3 Experiments

We performed computer experiments to demonstrate the performance of our algorithm and its corresponding annotation. All experiments involve multiple

genes that are pair-wise analyzed for frameshift overlap. We use the description extraction algorithm from Chapter 3, but we report only on frameshift variants. In all experiments where multiple genes are pair-wise analyzed we use *Bonferroni correction* for multiple testing with the significance threshold:

$$1 - \frac{\alpha}{m},$$

where m is the number of tests that are performed and α is fixed at 0.05. This gives us a conservative bound on the statistical significance. For a pair-wise analysis with n genes we typically do $n^2 - n$ tests. Although the Bonferroni correction is too restrictive, we consider a full analysis of multiple testing on this data beyond the scope of this chapter.

4.3.1 Intra-species frameshifts in *E. coli K-12*

In this experiment we analyze $4{,}305$ genes[1] pair-wise of *E. coli K-12* taken from Uniprot [UniProt Consortium, 2015]. We are interested in the self overlap (in terms of frameshifts) of this species. For this species we obtained a codon frequency table from [Nakamura et al., 2000] and we used this additional information for the probability calculation as described in Section 4.2.1. After Bonferroni correction, we selected every pair that contained at least one statistically significant frameshift variant.

The average gene length (on the protein level) is approximately 315 amino acids. The total computation time was around 48 hours on a desktop PC (3.4 GHz and 16 GB RAM) using a single thread. Which results in an average extraction time of 0.01 per pair of genes. This process could trivially be parallelized.

Out of the $18{,}528{,}720$ pair-wise tests, 245 pairs had a significant frameshift overlap. Within these pairs 359 unique proteins are present. In principle the frameshift overlap property is symmetrical, however, it is often possible to extend a frameshift with one amino acid in one of the directions. In such a case the added amino acid has the same frameshift overlap by chance. Often we see symmetrical frameshift overlaps with a length difference of one. For

[1] Not all of the annotated protein sequences encode for a gene.

large frameshifts this is usually not a problem in terms of its significance, but smaller frameshifts can be selected as being significant by adding one amino acid. This accounts for the odd number of significant pairs as well as the larger than expected number of unique proteins. In Table 4.3 the top 10 frameshift overlaps are given.

Table 4.3: The 10 most significant frameshift overlaps in *E. coli K-12*. Only unique pairs are given. For all these pairs the corresponding symmetrical overlap is also present and significant. The calculated numerical significance is 1 for all of these pairs.

Protein	Protein	Type	Length
P75617_YAAW	P28697_HTGA	inverse +2	195
Q9Z3A0_YJGW	P39349_YJGX	inverse	84
P0AE48_YTFP	P08339_Y4223	inverse +1	84
P76158_RZPQ	C1P601_RZOQ	+2	83
P27838_CYAY	P11291_Y3808	inverse	68
P77551_RZPR	P58042_RZOR	+2	61
P75719_RZPD	P58041_RZOD	+2	59
P75712_ALLP	P75711_YBBV	+2	45
Q47536_YAIP	P75697_YAIX	+2	42
P76066_YDAW	P77551_RZPR	+1	36

The longest significant result in Table 4.3, the pair P75617_YAAW and P28697_HTGA, is described as an overprinting event in [Delaye et al., 2008]. In this case, the protein P28697_HTGA forms a complete frameshift overlap with part of the P75617_YAAW protein. Another striking observation is the relative long lengths of the frameshift overlaps especially when taking into account the corresponding DNA sequences, given the expected length of longest common substring between two random strings [Abbasi, 1997]. The technique for finding frameshifts can be applied to initiate further research like [Delaye et al., 2008] aimed towards finding evidence for the evolution of novel genes. The names containing RZ seem to be somewhat related genes, however, although

the selected pairs show a strong significance, no other combinations have a significant frameshift overlap, suggesting that frameshift variants are an important evolutionary mechanism. Finally, from the data from the complete experiment (data not shown in Table 4.3), we observe that almost all pairs of proteins show a longer than expected frameshift overlap; around 15 amino acids on average.

The significant proteins were used to check whether any clustering of frameshift overlaps could be found. We calculated a 359×359 distance matrix, where distance is defined as $d(r, s) = r_{\text{fs}} / \max(|r|, |s|)$, with r_{fs} the length of the longest frameshift variant. It turns out that almost all of the frameshift overlaps occur in pairs. And, in contrast to what the RZ proteins in Table 4.3 might suggest, there are no clusters to be found.

4.3.2 Inter-species frameshifts between *E. coli K-12* and *S. enterica*

The results from the experiment in Section 4.3.1 show a strong evidence for intra-species frameshift overlaps. In this experiment we investigate whether the same evidence can be found inter-species. We use the same data for *E. coli* as in Section 4.3.1 and we added $4{,}533$ protein sequences from *S. enterica* taken from Uniprot [UniProt Consortium, 2015]. Instead of a full pair-wise comparison we calculated the maximum frameshift overlap for each of the $4{,}305$ *E. coli* proteins with any of the $4{,}533$ *S. enterica* proteins. For the frameshift probability calculation we used the same codon probabilities for *E. coli* as in Section 4.3.1. No such data was available for *S. enterica*, so we used a slightly less accurate probability calculation based on the probabilities of the occurrence of the amino acids (taken from the actual protein sequences). The top 10 frameshift overlaps are given in Table 4.4. The significance of the frameshift overlaps is given as $1 - P$, where P is the probability of the frameshift.

Although the results are not as striking as in Section 4.3.1, there are still significant inter-species frameshift overlaps. Again, this method can be used to identify promising candidates for further evolutional research.

From the data from the complete experiment (data not shown in Table 4.4),

Table 4.4: The 10 most significant frameshift overlaps between *E. coli K-12 and* S. enterica. Only unique pairs are given. For all these pairs the corresponding symmetrical overlap is also present and significant. The significance is given as $1 - P$ in order to show some meaningful value.

E. coli	*S. enterica*	Type	Significance	Length
P08339_Y4223	Q7CP87_Q7CP87	inverse $+1$	$1.06\text{e}{-42}$	59
P75712_ALLP	Q8ZR81_Q8ZR81	$+2$	$5.83\text{e}{-30}$	34
P0CF79_INSF1	Q8ZRJ4_Q8ZRJ4	inverse	$6.75\text{e}{-27}$	30
P11291_Y3808	P56978_CYAY	inverse	$3.69\text{e}{-24}$	28
Q79CP2_YGIA	Q7CPS1_YGIB	$+2$	$6.62\text{e}{-21}$	27
P0ADP5_YIGM	P0A2Q4_MESALTY	inverse $+1$	$3.12\text{e}{-19}$	27
P0ACW2_YDBJ	Q8ZP90_Q8ZP90	inverse $+1$	$3.15\text{e}{-19}$	26
P0AC96_GNTU	Q8ZLG4_Q8ZLG4	inverse	$4.32\text{e}{-23}$	26
P76323_INTG	P26462_FLIE	inverse	$1.73\text{e}{-22}$	25
P58042_RZOR	Q8ZQ98_Q8ZQ98	$+1$	$3.11\text{e}{-21}$	25

only 16 protein sequences from *E. coli* do not have any non-trivial frameshift overlap with any other protein sequence of *S. enterica*. The shortest non-trivial overlap is of length 7. The average length of frameshift overlaps is slightly less than in our previous experiment; around 12 amino acids.

4.3.3 Quality of the frameshift annotations

In this experiment we look at the quality of the reported frameshifts. We selected 578 DNA variants reported in the LOVD DMD Database [Aartsma-Rus et al., 2006][2] in coding regions that lead to a (predicted) frameshift variant. For all of these variants we checked whether the frameshift variant reported by our method corresponds to the frameshift variant that can be inferred from the DNA variant. The variants can be classified into 4 disjunct sets: deletions with 394 variants, duplications with 194 variants, insertions with 19 variants, and deletions/insertions with 26 variants. Note that there are no complex frameshift variants present in this database as it contains mainly on simplex

[2] `https://www.dmd.nl/nmdb/home.php?select_db=DMD`

DNA variants. These variants can possibly be combined to construct a complex frameshift.

For this experiment we do not consider (predicted) frameshift variants with a length 1 or 2, because no reasonable annotation based on only the protein sequence can be expected. These amount to 15 cases in this dataset. In these cases often an ambiguous result is returned with our method. Usually, a combination of type $+1$ and type $+2$ is reported. One could argue that a frameshift with a length less than a certain length, depending on the input size, should not really be considered to be a frameshift. In either case it is in general impossible to distinguish such small frameshifts by looking only at the protein sequences, in particular frameshift variants of length 1 or 2.

Deletions

Of all 394 deletion variants, only 2 unexpected results were found: $\mathrm{c.980del}$ with length 8 and $\mathrm{c.3747del}$ with length 6. Both should be characterized as type $+2$, however type inverse $+1$ was reported. Interestingly, the average length of frameshift variants is around twice the length of the misreported variants.

Duplications

Of the 194 duplication variants we have 3 wrongly reported type inverse $+1$ frameshifts all resulting in frameshifts of length 6: $\mathrm{c.4634dup}$, $\mathrm{c.5697dup}$, and $\mathrm{c.6848dup}$. In addition, we have one variant for which no frameshift was detected: $\mathrm{c.1540dup}$, instead the trivial deletion/insertion was returned.

Insertions and deletion/insertions

A minority of the variants in the DMD database leading to a frameshift variant is either an insertion or a deletion/insertion. There are 2 mis-classified variants reported for insertions: $\mathrm{c.9672\_9673ins}$ with length 6, and $\mathrm{c.10406\_10407ins}$ with length 4. Again, inversions were reported.

Overall, 7 of the 578 generated frameshifts seem to be unexpected when taking

the DNA variant into account. All of these frameshifts have a length of 8 amino acids or less, which is much less than the average frameshift length (21) of all variants in the LOVD DMD Database. Note that in all cases the reported frameshift annotation was not wrong (on the protein level), just unexpected from the inference of the DNA variants.

4.4 Discussion

In this section we explore the effects of the back-translation method described in Section 4.2.2 and we provide a suggestion for the extension of the HGVS nomenclature [den Dunnen et al., 2000] for the description of frameshift variants.

4.4.1 Back-translation

Here we elaborate on the back-translation method at its effects on real-life data. When considering a naive back-translation, i.e., no additional information available, a striking pattern is observed when measuring the ambiguity, i.e., the number of codons that represent a particular amino acid, per amino acid, see Figure 4.2.

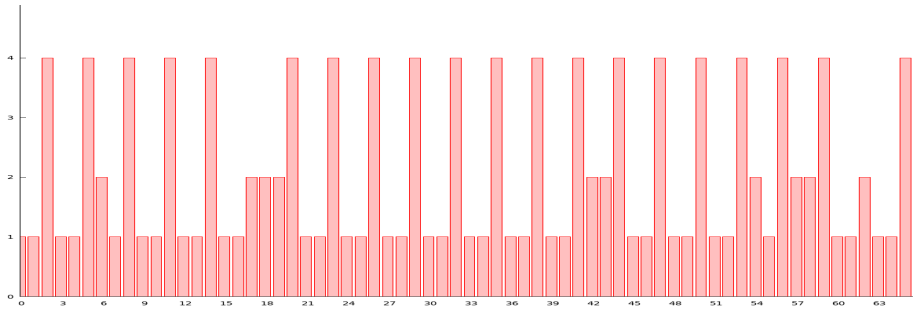


Figure 4.2: The ambiguity of the first 66 nucleotides of protein P28697_HTGA of *E. coli*. The first two nucleotides of each codon typically have a low ambiguity (one or two), while the last nucleotide has a high ambiguity.

In Table 4.5 we show the average ambiguity for each nucleotide on each of the three positions in a codon using the five different frameshift types. We selected

the most significant frameshift for the intra-species frameshifts overlaps in *E. coli*. Note that for the naive method the selection should not make any difference. The different types of frameshift show no significant variation in the distribution of the ambiguity.

Table 4.5: The ambiguity per nucleotide position in a codon.

Method	Nucleotide 1	Nucleotide 2	Nucleotide 3
Naive	1.25	1.07	3.15
Frameshift	1.10	1.01	1.06

The results in Table 4.5 show that back-translation method, explained in Section 4.2.2, using frameshift information results in a near perfect prediction of the individual nucleotides.

4.4.2 Proposed HGVS Descriptions

The HGVS nomenclature [den Dunnen et al., 2000] dealing with protein descriptions is, in our opinion, less well defined and structured as its DNA counter part. Especially so for the description of frameshift variants. Our main objection is the fact that in the current standard, information is lost when describing a frameshift variant because only the length of the frameshift is included in the description, but the actual amino acid sequence is omitted, e.g., $\mathrm{p.Arg97ProfsTer23}$. Furthermore, all description regarding protein sequences are rather verbose using the 3-letter abbreviations for amino acids (including capitalization). This makes human interpretation difficult, e.g., $\mathrm{p.Lys2\_Met3insGlnSerLys}$, especially since most of the HGVS operators are also described by a 3-letter symbol (del, ins, etc.).

Currently, only two types of frameshift can be described with the HGVS nomenclature: $+1$ and $+2$, but they are not distinguished, and there is no way of describing a complex frameshift variant.

In order to accommodate for more comprehensive and complex frameshift variants, we propose the following structure and properties for protein descriptions:

- all protein sequences are described using the single letter codes for amino acids;
- positions are not prefixed with the (redundant) reference amino acid (c.f. DNA descriptions);
- the same basic operators are used as for DNA description with the exception of inversion;
- frameshift descriptions are given as deletions/insertions with extra annotation so that no information is lost when describing a protein sequence.

Given a HGVS frameshift description using the current standard: $\mathrm{p.Cys10Valfs*16}$ can be written as: $\mathrm{q.10\_3685delins10\_24}$`VMKEKMFKRKHSQNG`$|2$. In this format the transposition notation from Chapter 3 is used to denote the frameshifts reference coordinates; the reference sequence from position 10 until position 3685 is replaced by the reference sequence from position 10 until position 24 with a frameshift of type $+2$ denoted by the operator $|$. These operators can be chained for the annotation of a multi-type frameshift variant. The actual amino acid sequence is also included (`VMKEKMFKRKHSQNG`) as in general many of such sequences could exist.

For more complex frameshift variants we can use to full transposition notation for alleles: $\mathrm{q.10\_3685delins}$[`QSHK`$;10\_15$`KEKFMS`$|1|2;$`AAA`]. In this case some parts of the inserted sequence could not be (effectively) described as frameshifts. A frameshift variant is described as 10_15`KEKFMS`$|1|2$. In this case this is a multi-type frameshift variant; type $+1$ and type $+2$. Note that the frame numbering of the NCBI is different: both frame 0 and 1 denote the 'normal' reading frame, 2 corresponds to the second base offset, and 3 corresponds to the third base offset. This numbering is quite different from the current HGVS nomenclature. At this point it is unclear how to harmonize the different numberings.

The proposed format is implemented in the Description Extractor package for Mutalyzer [Wildeman et al., 2008] (see Chapter 3) and is available at: `https://github.com/mutalyzer/description-extractor`.

4.5 Conclusions

We introduced a method for efficiently finding different types of frameshift variants in protein sequences without using any DNA level information. This method has as additional advantage that it can calculate the probability of a frameshift on several different levels depending sometimes on additional information available. It can use any codon table as basis, and the frameshift calculations can be precomputed for such a codon table resulting in efficient computation of the description and annotation.

This method can also be effectively used to generate a back-translation to DNA sequences, both for the reference sequence and for observed sequence.

We have suggested that large frameshifts are relatively common intra-species within *E. coli* and to a lesser extent inter-species. This method is able to identify promising candidates for evolutional research.

The generated frameshift descriptions on a real-life LOVD database show that the quality of the reported descriptions is high with only few unexpected results for very short length frameshifts.

The HGVS nomenclature has to be updated to accommodate for these new (complex) frameshift variants.

Chapter 5

Extraction of HGVS Variant Description for Short Tandem Repeat Structures

Characterization of complex variants such as short tandem repeats (STRs) is an important tool in forensics. Historically, these variants are difficult to characterize. Here, we propose a method that can automatically construct a description following the philosophy of the standard nomenclature of the Human Genome Variation Society (HGVS). Within these descriptions there is also a way of describing variation in the flanking regions relative to the repeat structure.

We accurately detect repeat units in a (reference) sequence and we can use these repeat units to generate a meaningful reference-based description. We propose an alternative repeat variant that can be added to the existing HGVS nomenclature and finally, we provide an implementation of our proposed methods as part of the Description Extractor package for Mutalyzer: `https://pypi.python.org/pypi/description-extractor`. The Python source code is available at: `https://github.com/mutalyzer/description-extractor`.

5.1 Introduction

A considerable part of the genome consists of repeated regions that occur in many forms. In this chapter we focus on a particular from of repetition in genomic sequences called *Short Tandem Repeats* (STRs) also known as microsatellites or simple sequence repeats. STRs contain small motifs or *units* ranging from 2–5 base pairs which are repeated in *tandem*, i.e., immediately adjacent, typically 5–50 times. The combination of the lengths and different loci of STRs is a powerful tool in population genetics and forensics. The analysis of repeated sections of the genome have always been problematic not only on the sequencing level, but also in automated analysis [Weischenfeldt et al., 2013]. In this chapter we present a method for the automatic construction of descriptions of repeated structures, especially tailored for forensic applications. These descriptions are based on the recommendations of the Human Genome Variation Society (HGVS) [den Dunnen et al., 2000, Taschner and den Dunnen, 2011]. While the HGVS nomenclature currently provides a way to describe one repeat unit in tandem, forensic applications demand a more rigorous construct to describe the repeated structures, possibly containing multiple repeat units, as a whole. In any case, the detection of repeated sequences including the repeat unit is a non-trivial task.

The STR allele descriptions need to maintain compatibility with existing databases including the definition of the repeat units, annotation on the forward or reverse strand, etc. In Section 5.3 of [Gettings et al., 2015] an attempt is made to create a uniform representation of commonly used STR alleles in terms of strands and repeat units. They also envision three possible methods for describing STR alleles:

1. complete sequence strings;
2. a bracketed description;
3. unique identifiers.

A description method based on the use of complete sequence strings entails the direct use of Next Generation Sequencing output. This output is deemed to be

large and complex and therefore not necessary tailored to the intended purpose here. In particular, storage in databases will become more costly and matching techniques is expected to require more computational power. On the other hand, more relevant data can be stored; small variants in the "flanking" region are also present in this approach. The bracketed description is often informally used in reports and is especially useful in describing the actual repeat units and the number of occurrences, e.g., TCTA[9], where the unit TCTA is repeated 9 times. Having this kind of descriptions means that comparison on (allele) length is relatively straightforward. However, combinations with changes in the flanking regions are more difficult to express as no real guidelines exist at the moment. Finally, one could consider unique identifiers for each STR allele. In [Gettings et al., 2015] it is suggested that a somewhat meaningful identifier can be assigned. While based on the number of repeats and its flanking variations these identifiers give no proper description. Unique identifiers will of course be superior when performing database searches. However, this approach requires more administration as these identifiers need to be globally unique. Furthermore, humans (in a laboratory setting) can have difficulties in dealing with these kind of descriptions. In particular, partial searches are impossible.

Other attempts of automatically characterizing STR alleles have been made. Most importantly in [Anvar et al., 2014] where a sequence can be matched against a pre-defined database of STR allele descriptions given as regular expressions. Some small variants in the flanking regions are tolerated, but no part of the description. Amongst other reasons, this method does not result in a full description of the observed sequence. In [Hoogenboom et al., 2016] the results from [Anvar et al., 2014] are used and improved upon by focusing on the reduction of PCR stutter noise. An accurate estimate of repeat lengths can be given and there is some support for the characterization of unknown STR alleles.

In this chapter we propose a method for the automatic generation of reference based descriptions for STR alleles. In essence this is closest to the aforementioned bracketed approach. Apart from giving descriptions for the repeat structure, we also incorporate the classical extraction method from Chapter 3 for the description of the flanking, and possibly interspersing, regions

in such a way that the variation within these flanking regions is described relative to the repeat structure, cf. the descriptions of coding regions in the HGVS nomenclature. This approach combines the strengths of both methods while eliminating their respective drawbacks.

The remainder of this chapter is structured as follows. In Section 5.2 we present a method for a reference sequence based description for STR alleles. This method is in principle database-free, but can also be used in combination with a DNA data bank yielding descriptions that are closer to the annotation used in the literature. In Section 5.3 we introduce the data and experiments, followed by a discussion of the results of these experiments in Section 5.4 and the conclusions in Section 5.5.

5.2 Methods

First we focus on the detection (or extraction) of repeat units from a string. Later we combine these results and an adaptation of the HGVS extraction algorithm in [Vis et al., 2015] to generate a reference-based HGVS-like description. Finally, we address the problem of generating descriptions relative to the repeat structure following a similar approach as is used in the HGVS nomenclature when describing variants in the $5'$ and $3'$ UTR relative to the coding region of a transcript.

5.2.1 Finding repeat units

In order to have a database-free method we must be able to find candidate repeat units in a string. For the concrete application at hand, STRs, we can take advantage of the properties of STRs. The *tandem*, i.e., immediately adjacent, property of these repeats is of primary concern. Often suffix trees are used for finding substrings with certain properties. However, as we deal with non-overlapping directly adjacent repeats, a more straightforward method can be used; a variation on classical run-length encoding [Golomb, 1966]. In the classical run-length encoding algorithm, the length of the repeat unit is fixed at 1. The algorithm can easily be extended to arbitrary fixed-length repeat

units. For this particular application we do not know the length of the repeats units for a certain string a-priori. As the repeat units within STRs are of limited length, i.e., 2–5, one can easily try to find repeat units of all of these sizes. Special care has to be taken when dealing with self-similar repeat units, e.g., `TATA`, because an other, smaller unit exists, `TA` describing the same repeated sequence. In these cases we are interested in the smallest unit describing the repeated sequence. Algorithm 1 is an implementation of such a variation on the run-length encoding algorithm.

In Algorithm 1, we iterate over the length of the string (line 3). Every candidate length of the repeat unit (k) is tried for the given position in the string (line 5). Candidates are limited by the optional parameters min_length and max_length. The number of repeats for this particular unit is tracked (line 11). If for the given repeat unit length we obtain a larger covered region in the string, this is considered the "best" candidate repeat unit at this position in the string (line 13). Ultimately, the selected repeat unit for this position is added to the repeat unit set (line 17), and the next possible position for a repeat unit is considered. In the end we can end up with a repeated region at the end of the string, so we need to add that as necessary (line 21). The algorithm returns a partitioning of the string in repeat units.

On the resulting partitioning it is trivial to filter for a specific minimum or maximum number of occurrences, as well as constructing a set of unique repeat units with characterized lengths and counts. Such a set can be used in the second stage of our method.

An implementation in Python of our proposed method is provided at: `https://github.com/mutalyzer/ssr-extractor`.

5.2.2 Reference-based description of the repeat structure

The purpose of this method is to yield a bracketed type of description, similar to the existing HGVS nomenclature for repeated sequences, given a set of repeat units. This set could be generated by the application of Algorithm 1, or alternatively, a set from a database or literature can be used. As further input the method expects a reference string as well as the observed string.

Algorithm 1 SHORTTANDEMREPEAT

input: $string, [min\_length = 2, max\_length = 5]$ (optional unit length)
output: $repeats$ (partitioning in repeat units)

while $0 \leq i < |string|$ **do**
  $max\_count = 0, max\_k = 1$
  for $k \in [min\_length, \ldots, max\_length]$ **do**
    $count = 0$
    for $i + k \leq j \leq |string| - k + 1$ **step** k **do**
      if $string_{i \ldots i+k} \neq string_{j \ldots j+k}$ **then**
        break
      end if
      $count \leftarrow count + 1$
    end for
    if $count > 0$ **and** $count \geq max\_count$ **then**
      $max\_count \leftarrow count, max\_k = k$
    end if
  end for
  $repeats$.add($\{i, max\_k, max\_count\}$)
  $i \leftarrow i + max\_k(max\_count + 1)$
end while
if $max\_count > 0$ **then**
  $repeats$.add($\{i, max\_k, max\_count\}$)
end if

return $repeats$

As we try to follow the current nomenclature standard closely, the resulting textual output of these descriptions will be resembling the HGVS nomenclature. However, we introduce a new type of variant, for describing repeated sequences, with slightly stricter defined semantics. As none of this is part of the official HGVS nomenclature we will, for now, use a different notation to avoid confusion. The newly introduced type of variant is a form of the deletion/insertion operator in HGVS. For its notation we list the repeat unit

(in bases) once, followed by the number of its consecutive occurrence in the observed sequence in parentheses, e.g., ATCT(4). This denotes that at a certain position in the reference sequence all tandem occurrences of this particular repeat unit are deleted and the specified number of occurrences is inserted in the observed sequence, for the given example, 4.
Consider the example:

$$\begin{aligned} R &= \texttt{ATCTATCTATCTGGATCT} \\ S &= \texttt{ATCTATCTATCTGGATCTATCT}, \end{aligned}$$

where R is the reference sequence and S the observed sequence and we choose {ATCT} as repeat unit set. The aim is to come to a description that shows that the repeat unit is present 4 times in the observed sequence. The repeat unit is present only 3 times in the reference sequence. The remainder of both sequences is a complete match. We propose to describe this STR allele as: [ATCT(4);13_14;ATCT(2)], giving an unambiguous description from the reference sequence to the observed sequence. Note that the identical (matching) part of the reference sequence is described using our transposition notation defined in Section 3.1.1. Indeed, the whole description should be interpreted as a deletion of the complete reference string with the observed string inserted expressed as transpositions of the reference string and repeat units. Small variants, e.g., single nucleotide substitutions can be described in the classical way. One could remark that the repeat unit itself can also be expressed as a transposition, however, in order to be closer to both the currently used notation in databases as well as the HGVS nomenclature, we prefer the notation in bases for the repeat unit. Furthermore, it might be required to give a description of an STR allele where the repeat unit is not present in the reference sequence, in which case there is no valid transposition.

The extraction of the description follows the method described in Section 3.2 with a slight alteration; we introduce a so-called masking character, i.e., a character not part of any of the molecular alphabets with the added property that this masking character does not match any character (including itself). We denote this character by $. The first step of our method masks all the occurrences of the repeats from the repeat unit set with the masking characters

in both the reference and the observed sequence. In this way we can abstract from the actual sequence without losing the positioning information which is needed to construct the description. Following the aforementioned example R and S are transformed into `GG$$$$$$$$$$$$G` and `GG$$$$$$$$$$$$$$$$G` respectively. These masked strings are used as input for the variant description extraction (VDE) method presented in Chapter 3. In this particular case, the variant extraction does not yield any variants as both prefixes and suffixes are identical, but as mentioned before, small variants can be described in the classical way.

Regular substring matching is used to fill in the masked regions with repeat units from the repeat unit set, e.g., at position 3 of the observed sequence we find 4 occurrences of the unit `ATCT`. The descriptions resulting from the VDE and the masked regions are then merged, where identical regions are transformed into transpositions. When dealing with multiple variants, the VDE will sometimes combine multiple atomic variants into larger deletions/insertions based on the description length of the HGVS nomenclature. In particular instances it may happen that variants are combined while spanning a repeat unit as well. In these cases we have to split the combined variant and interleave it with the repeat variant in order to perform the merging step.

5.2.3 Relative description of the flanking regions

The final part of our method deals with the automatic generation of relative descriptions, i.e., positioning relative to some part of the reference sequence, for describing variation in the flanking regions. These relative descriptions are similar to the coding region oriented descriptions used in HGVS for the description of $3'$ and $5'$ UTR variants, e.g., `c.-56C>T` and `c.*32G>A`. As the variation in the flanking regions can be characteristic for certain STR alleles (in combination with the repeat structure) this is an important element of the STR allele description. A naive approach could be generating a description that is a composite deletion/insertion, containing transpositions and insertions, of the whole reference sequence, including the flanking regions. However, the resulting descriptions can become quite complex and difficult to interpret,

especially when variable length flanking regions are used. Furthermore, these descriptions distract the attention from the actual repeat structure, making it more difficult to identify. In our method we propose to solve this problem by using a relative description, i.e., we describe the variation in the flanking regions relative to a predefined region. In practice, we use the repeat structure, but in general this can be any substring of the reference sequence. A uniform way of defining the repeat structure would be from the first occurrence of any of the repeat units in the repeat unit set to the last occurrence of any of the repeat units. This results, when applying the steps in Section 5.2.2, in descriptions that look like `[-15A>T;ATCT(4);*4_5insAT]`, cf. the HGVS nomenclature by using the minus operator to describe variation before the repeat structure and the star operator to describe variation after the repeat structure.

The Mutalyzer tool suite [Wildeman et al., 2008] module that deals with the generation of relative descriptions is called the Crossmapper. This module performs the non-trivial conversion between the different positioning schemes in HGVS. In particular it is used to convert variant descriptions on different transcripts, i.e., from genome to transcript and vice versa. We can make use of this module for our purpose by introducing an artificial coding region that represents the repeat structure. After generating a naive (genomic) description of the complete reference sequence, we can apply the Crossmapper, given this artificial transcript, providing us with a view relative to the repeat structure, e.g., `TPOX(122_142):[-50G>T;TGAA(6)]`, where the selected transcript is made explicit in the notation of the reference sequence. In this example, the repeat structure is selected from position 122 to 142. In the flanking region there is a single nucleotide substitution 50 bases before the start of the repeat structure. The flanking region after the repeat structure contains no variants in this case.

5.3 Experiments

In this section we describe the experiments designed to illustrate the performance of our proposed method for the generation of STR allele descriptions. We focus on forensic use cases. In Table 5.1 we introduce a set of 24 STR loci that are commonly used with their repeat structure annotation as found

in [Gettings et al., 2015]. For each locus only one STR allele is annotated, which can serve as the reference sequence for that locus. We have a dataset containing 1,631 STR alleles from the loci in Table 5.1, including the reference sequence. In Table 5.2 we give a characterization of the dataset.

Table 5.1: A set of 24 frequently used STR loci with their respective repeat structure. All repeat structure characterizations are normalized following [Gettings et al., 2015], but do not include any flanking regions.

STR locus	Repeat structure
Amel[†]	none
CSF1P0	AGAT(13)
D10S1248	GGAA(13)
D12S391	AGAT(11) AGAC(7) AGAT
D13S317	TATC(11) AATC(2)
D16S539	GATA(11)
D18S51	AGAA(18)
D19S433	AAGG AAAG AAGG TAGG AAGG(12)
D1S1656	TAGA(16) TAGG TG(5)
D21S11	TCTA(4) TCTG(6) TCTA(3) TCA TCAT(2) TCCATA TCTA(11)
D22S1045	ATT(14) ACT ATT(2)
D2S1338	TGCC(7) TTCC(13) GTCC TTCC(2)
D2S441	TCTA(12)
D3S1358	TCTA TCTG TCTA(14)
D5S818	AGAT(11)
D7S820	GATA(13)
D8S1179	TCTA TCTG TCTA(11)
DYS391	TCTA(11)
FGA	TTTC(3) TTTT TTCT CTTT(14) CTCC TTCC(2)
PentaD	AAAGA(13)
PentaE	AAAGA(5)
TH01	AATG(7)
TPOX	AATG(8)
vWA	TCTA TCTG(5) TCTA(11) TCCA TCTA

[†] Amelogenin (Amel) is not an STR, but often included as a way to determine sex.

The automatic extraction of the repeat unit set in Table 5.2 is performed by applying Algorithm 1 on each of the STR alleles belonging to a certain STR locus. An aggregated repeat unit set is then created for each locus by only selecting the repeat units that are present in all separate repeat unit sets. In addition, the total count for each repeat unit should be more than 5. This selection is rather arbitrary, but in practice it yields reasonable repeat unit sets. Although a comprehensive method of finding the "best" repeat unit set given a set of STR alleles is outside the scope of this chapter, some improvements are discussed in Section 5.4.

As can be observed from Table 5.2, the majority of the repeat units is of length 4. Two repeat units are of length 5, one of length 2 and one of length 3. The automatically extracted repeat units correspond in most cases to the ones that are described in literature, especially when taking into account the possible rotations of the pattern (i.e., `ATCT` can also be described as `TCTA`, `CTAT` and `TATC`). In multiple cases, i.e, CSF1P0, D13S317, D21S11, D8S1179 and FGA, two rotations of the same repeat unit are automatically extracted (usually `ATCT`), this phenomenon usually arises when the same repeat unit occurs in multiple, separate, repeated regions, where the surrounding bases have an influence on the selection of the particular rotation. In some cases, i.e., D13S317, D16S539 and D7S820, a novel repeat unit is extracted. This unit is present in the STR allele, but it may not occur in a variable number over the STRs, i.e., they occur always a fixed number of times. This is not a property that can be intrinsically detected by Algorithm 1. For one locus, TH01, our dataset clearly contains the opposite strand from the one that is commonly used in literature.

As a second experiment we applied the full description generator method on the full dataset from Table 5.2, once using the automatically extracted repeat units and once using the repeat units taken from [Gettings et al., 2015] (where we used the reverse complement for TH01). As reference sequence we selected the STR allele with the repeat structure from Table 5.1. Note that even when we extract a STR description for the reference sequence itself, our method gives a description in terms of repeat units, where, on the other hand, the VDE would result in the idem description. The repeat structure location

Table 5.2: Characterization of the dataset used for the experiments. For each STR locus the number of variant alleles is given as well as their annotated repeat unit set (from [Gettings et al., 2015]) and the automatic extracted repeat unit set.

STR locus	Count	Repeat unit set (literature)	Repeat unit set (extracted)
Amel	3	none	none
CSF1P0	22	{AGAT}	{TAGA, AGAT}
D10S1248	16	{GGAA}	{GGAA}
D12S391	107	{AGAT, AGAC}	{CAGA, TAGA}
D13S317	81	{TATC}	{AATC, ATCT, TATC}
D16S539	43	{GATA}	{ACAG, GATA}
D18S51	41	{GAAA}	{AGAA}
D19S433	56	{AAGG}	{AGGA}
D1S1656	76	{TAGA, TG}	{GT, TAGA}
D21S11	323	{TCTA, TCTG}	{TCTG, TATC, TCTA}
D22S1045	34	{ATT}	{ATT}
D2S1338	176	{TGCC, TTCC}	{TGCC, TTCC}
D2S441	28	{TCTA}	{TCTA}
D3S1358	48	{TCTA}	{TGTC, TATC}
D5S818	60	{AGAT}	{AGAT}
D7S820	112	{GATA}	{GATA, TTT}
D8S1179	66	{TCTA}	{TCTA, TATC}
DYS391	13	{TCTA}	{TATC}
FGA	69	{TTTC, CTTT, TTCC}	{TCCT, TTTC, TCTT}
PentaD	49	{AAAGA}	{AAGAA}
PentaE	36	{AAAGA}	{AAAGA}
TH01	32	{AATG}	{GTCT, ATCT, ATCC}†
TPOX	26	{AATG}	{TGAA}
vWA	114	{TCTG, TCTA}	{TATC}

† Our sequence data contains the opposite strand from the one that is used in literature.

is automatically designated by the method described in Section 5.2.3. In the general case the start and end positions of the repeat structure could be chosen

based on literature, however, currently there is no consensus on these positions.

When comparing the generated descriptions using both versions of the repeat unit set, we observe that in the majority of cases either the descriptions are identical (when the repeat unit set was identical) or highly similar, especially when considering the length of the description, or the total number of repeat units covered in repeated sequences, e.g.:

```
D19S433(37_75)  :  [AAGG(1);5_6;AAGG(1);11_14;AAGG(6)]
D19S433(38_83)  :  [AGGA(1);5_10;AGGA(7);39_41;AGGA(1)],
```

where the same sequence is described by two different rotations of the same repeat unit: For the top description `AAGG` from literature is used, while `AGGA` is automatically extracted using Algorithm 1 and is used for the bottom description. The absolute length of both descriptions is very similar; the top one is one character shorter. However, the number of repeat units covered by the respective descriptions is in favor of the bottom description, where 9 units are covered as opposed to 8 in the top description. Both descriptions obviously yield the same observed sequence, but this might not be immediately clear. The location of the repeat structure is slightly different for both and so are the descriptions of this structure, not only the repeat unit itself (which is clear), but, more importantly, the interleaving transpositions. When storing these descriptions in databases unambiguous descriptions might be preferred.

5.4 Discussion

In this section we discuss some of the issues of the application of our method for the automatic generation of descriptions for STR alleles. In particular the usefulness of these descriptions in forensics.

5.4.1 Reference sequence

When the goal is to have easily comparable and meaningful descriptions without the need of resorting to tools, there must be a fixed reference sequence for each STR locus. As this kind of information is stored for relatively long times,

these reference sequences have to be invariant over time. General reference sequences deal with the whole genome, whole chromosomes or genes. None of these is particularly useful; the interesting regions are very small compared to chromosomes and usually not situated inside or near genes. We advocate the use of so-called Locus Reference Genomic (LRG) reference files [MacArthur et al., 2014]. LRGs are fixed references that do not change over time, furthermore, they can contain transcript annotation, which can be used for the specification of the location of the repeat structure. This has the additional advantage that the annotation of the repeat structure is explicitly present in the reference sequence. LRGs can be easily requested via the webpage: `http://www.lrg-sequence.org/`.

5.4.2 Repeat unit set per STR locus

After choosing a suitable reference sequence, there should also be a consensus on the repeat unit set that is used for a specific STR locus. Note that Algorithm 1 does not necessarily give the same repeat unit set for each allele of the same STR locus. This can be regarded as a potential problem for having unambiguous descriptions. Unfortunately, it is impossible to explicitly include this information in an LRG reference sequence. This means that either an additional database has to be curated, or an extension of the LRG sequences should be proposed containing the repeat unit set for each STR locus.

As we observed from the experiments in Section 5.3, repeat units can be rotated. The chosen rotation in literature seems to be, to a certain extent, arbitrary, e.g., the "best" fitting rotation for a certain (reference) allele. In the case of the FGA locus, the literature gives two rotations of the same repeat unit as part of the repeat unit set: `TTTC` and `CTTT`. Rotated versions of the same repeat unit occur more frequently when considering multiple STR loci. One could argue that, at least per locus, only one rotation of the same repeat unit can be present. On the other hand, considering that "best" fitting is rather ill-defined, an arbitrary selection could be made. e.g., the lexicographical smallest. This provides as an advantage that repeat structures over different loci can be easily compared.

The automatic selection of the repeat unit set used in Section 5.3 could be improved upon by disallowing rotations of the same repeat unit, as well as, disregarding any of the static repeat units, i.e., repeats that always occur a fixed number of times. Considering these improvements, the automatic extraction of the repeat unit set would result in the same set as is currently used in literature except for its rotations. Given the need for backwards compatibility we expect that no universal decision can be reached on the selection of the rotations. For this reason our approach has only a loose coupling between the automatic extraction of repeat units and the generation of the corresponding descriptions.

5.5 Conclusions

We introduced a method for the automatic generation of reference-based descriptions for STR alleles. Our method consists of two parts: 1) the automatic detection of a repeat unit set given a sequence and 2) the automatic generation of a reference based description given a repeat unit set, a reference sequence and an observed sequence. The automatic detection of repeat units uses a variation of the well-known run-length encoding algorithm and performs well on STR alleles with many small repeats. The results from this method can be used as input for the second part. Alternatively, repeat unit set defined in literature can be used.

The resulting descriptions are easily interpretable and can be made unambiguous for use in databases. We have discussed the application of our proposed method for a forensic use case with special attention regarding the unambiguity. We advocate the creation and use of specialized LRG reference sequence files.

Our proposed repeat variant, including its semantics and notation, should be proposed and added to the HGVS nomenclature.

Chapter 6

Meta-analysis of Disjoint Sets of Attributes in Large Cohort Studies

We will introduce the problem of classification in large cohort studies containing heterogeneous data. The data in a cohort study comes in separate groups, which can be turned on or off. Each group consists of data coming from one specific measurement instrument. We provide a "cross-sectional" investigation on this data to see the relative power of the different groups. We also propose a way of improving on the classification performance in individual cohort studies using other cohort studies by using an intuitive workflow approach.

6.1 Introduction

Cohort studies are frequently used in the biomedical field. The aim is to identify so-called risk factors correlated to a phenotype, usually a disease. A cohort is a group of people sharing a common characteristic during a certain period. Within this period some people either develop the studied phenotype or already exhibit it from the start. This subgroup is referred to as the *case* group, while the remainder of the people are designated as *controls*.

Nowadays a variety of different types of biomedical data can be gathered. Often physiological data such as gender, age, blood pressure and heart rate are present together with genomics data either consisting of the complete genomic sequence, but more often, in the form of *single nucleotide polymorphisms* (SNPs). While some diseases have a clear and simple genomic origin, many others are caused by a more complex combination of effects, therefore, data about the presence or concentration of all kinds of substances within the body such as *metabolites* are also important.

As new data generating methods become available during the period of a cohort study data is accumulated. Different types, e.g., genomics or metabolites, of data are collected from the same group of people resulting in more of less disjoint data sets describing a self-contained set of attributes of the same individual. It is unclear whether these data sets have the power to augment each other or whether they express the same knowledge. For example, due to a genomic defect a certain substance (metabolite) is under- or overexpressed.

The abundance of attributes per person results often in a skewed data set; a few instances versus a lot of attributes. As a consequence finding correlations in the data becomes tricky This effect is enhanced by the fact that widening (adding attributes) a data set is cheaper then adding more individuals. Which is even impossible at a later stage of the study.

Adding data is never free. This is especially the case in the biomedical field. The preparation and construction of these data sets are labor intensive and expensive in a financial sense. Furthermore, this process imposes a burden on the individuals from whom the samples must be taken. This results in a clear motivation to study the usefulness of the gathering of groups of additional

data.

In this chapter we investigate three cohort studies which each consists of a number of sets of attributes. In contrast to the more generally applied feature selection, we add whole sets of features instead of single features. We are primarily not so much interested in the absolute classification, but more in the added improvement on the classification of these separate sets. In addition, we propose a method of improving classification in one cohort study by adding data from different cohort studies.

The remainder of the chapter is organized as follows. In Section 6.2 we formalize a problem statement as well as introduce the cohort studies used in the experiments. Section 6.3 describes the workflow and tooling. We present the experiments in Section 6.4, and the conclusions to the study in Section 6.5.

6.2 Problem statement

In contrast to the classic feature selection problem, we have a number of disjoint sets of attributes that can either be included as a whole or they can be completely excluded from the study. Furthermore, we have a number of separate studies (concerning the same phenotype and covering the same sets of attributes) that can be pooled together in order to augment classifying power on a separate study. Given this setting, we define two meta-analysis problems:

1. Can we say something about the relative power of the (combinations of) sets of attributes?

2. Can data from different cohort studies be used to augment classifying power for a single study?

6.2.1 Anatomy of the data sets

As we are dealing with humans represented in these cohort studies, we must observe caution not to accidentally expose any details of these individuals nor are we authorized to publish any identifying details about the studies

themselves. Therefore we choose not to describe the actual cohort studies used, but we will give a feeling for the characteristics of these studies.

In the cohort studies we are interested in the classification of a certain phenotype: an aging-related disease. Earlier attempts to characterize this disease have resulted in a reasonably small (about 10) set of so-called risk factors which seem highly correlated with this disease. These risk factors are all physiological (e.g., age and sex), and, compared to many of the other attributes, easily gathered as they require minimal processing and analyzes.

In this study we will use three real cohort studies. The typical number of instances per study is between $1{,}000$ and $2{,}000$ of which approximately 25% is an identified case example (an individual exhibiting the disease).

6.2.2 Disjoint sets of attributes

The data in these cohort studies can be partitioned in disjoint sets of attributes describing a certain type of features. The first set we consider is the set of risk factors derived from earlier studies. We use this as a baseline for our study. Next, we have a set of several dozens of general physiological and behavioral characteristics. Both sets are rather easily constructed and therefore commonly present in similar studies. We have a set of genetic data containing several hundreds of SNPs, which are already selected as promising candidates for correlation regarding this disease. And finally, two sets of metabolitic data: concentration levels of several dozens of Free Fatty Acids (FFA), and about a hundred metabolites measured with NMR (nuclear magnetic resonance) spectroscopy, represented as areas under curve.

As the last three sets of data are considerably more difficult in terms of labor and finances to gather we are especially interested in their “added value” with regard to the general study and each other.

6.3 Workflows

To find answers to the problems stated in Section 6.2, we consider all possible combinations of the disjoint sets of attributes (except for selecting no data at

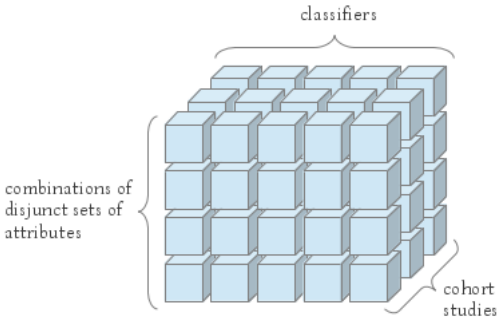


Figure 6.1: Schematic representation of the mining space. Each cube represents the classification power for the corresponding classifier on a combination of disjoint sets of attributes from a certain cohort study.

all) from all combinations of cohort studies. For each partition of the data we calculate the classification power of all classifiers, see Figure 6.1.

We propose an automated workflow for the calculation of all data points in Figure 6.1 and Figure 6.2. As a first approach all workflows are implemented as batch scripts using the pipes and filters design pattern. In particular, a selection of the standard Unix tools is used to perform the splitting of the respective data sets, the creation of folds, and data cleaning. We often augment the pipes and filters pattern [Gamma et al., 1995] by the data-driven pattern of the `make` utility. Commonly `make` is used to automatically build executables from source code, however it is not limited to building software. Indeed its data-driven paradigm combined with the power of declarative programming makes it especially useful in data centric workflows as described in [Robinson and Thain, 2013].

Although workflows have been introduced in the biomedical field, e.g., [Oinn et al., 2004], we observe that *ad hoc* solutions are frequently used preferring agile development over reusability and scalability. In our particular case, we want to combine data from separate data sets making the need for scalability more important. Furthermore we acknowledge that our first approach is highly technical and probably difficult to maintain in the biomedical

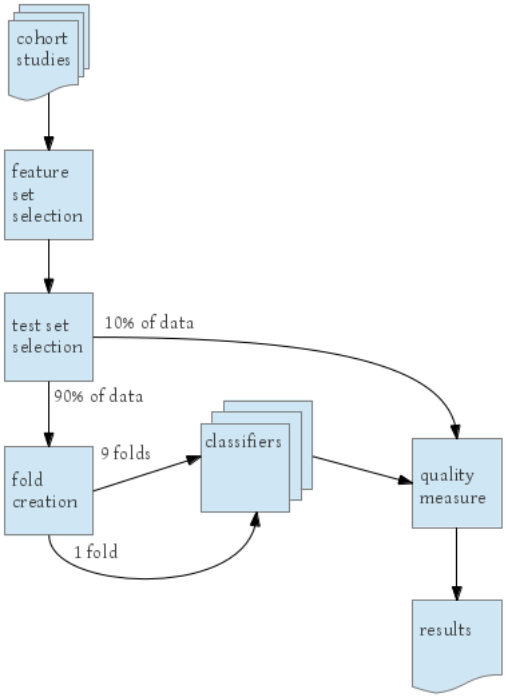


Figure 6.2: Graphical representation of the automated mining process. It shows the partitioning of the data and its distribution over the classifiers.

field. Most researchers in this field dealing with the data mining tasks on large cohort studies are not computer scientists or programmers. In order to enable them to conduct their independent research on these data sets, we have to provide them with high-level and easy to understand formal workflows.

To make our workflow accessible and reusable we introduced a more formal workflow. As most of our classifiers (see Section 6.3.1) are taken from the Weka toolbox [Hall et al., 2009], it seems natural to use the Weka KnowledgeFlow environment to design our workflows. Even though this environment is hardly a general workflow tool, our primary tasks are data mining related further advocating the use of this specialized toolkit.

To cater for a number of different recurring subtasks we designed four

separate workflows in the Weka KnowledgeFlow environment, see Figure 6.3:

(a) Data cleaning — In order to link several data sets together it is imperative that corrupt and inaccurate instances and attributes are detected and corrected;

(b) Data selection — Usually a single data set is used for training and validation purposes, here we deal with data from separate sets. This workflow enables the preselection of instances from these separate sets;

(c) Classifier training — This is the backbone of the actual data mining process. It is a fairly standard classifier training workflow;

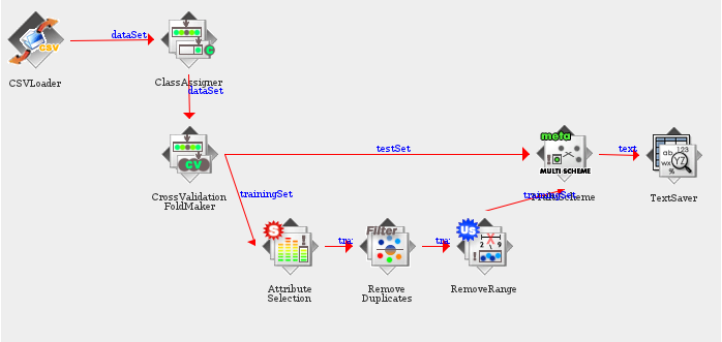
(d) Meta-learning — To effectively combine the results from the individual mining processes, we use the meta-learning workflow. It deals in particular with the hierarchical approach discussed in Section 6.4.4.

These separate workflows can in turn be combined into larger workflows if desired. Note that the meta-learning workflow is similar to the normal learning workflow, making the meta-level concept easier to understand, which is visualized in the actual workflows.

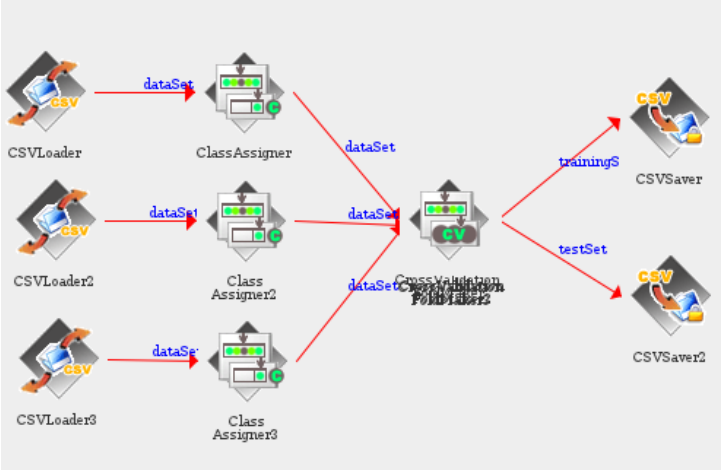
Although the Weka KnowledgeFlow environment might not be a full-scale workflow modelling language, it is well-suited for the analyses in this chapter. By using mostly classifiers from the Weka toolbox we avoid issues of incorporating different data mining tools. In a more general case it is often advantageous to use many different tools. Under this precondition it becomes difficult to solely use the Weka toolkit as a workflow modelling language and it is recommended to use a general workflow modelling framework like Taverna [Wolstencroft et al., 2013]. For this chapter we will consider this to be future work.

6.3.1 Classifiers

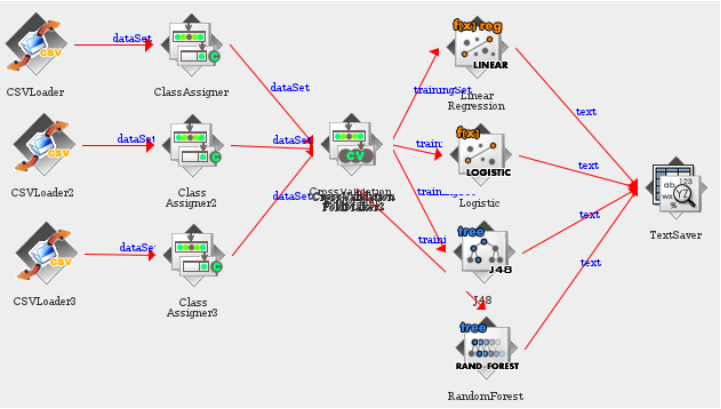
In our case we are interested in classification: to find causal attributes for the presence of a phenotype. The data at our disposal is highly heterogeneous in nature, therefore, we will use a variety of classifiers each especially suited



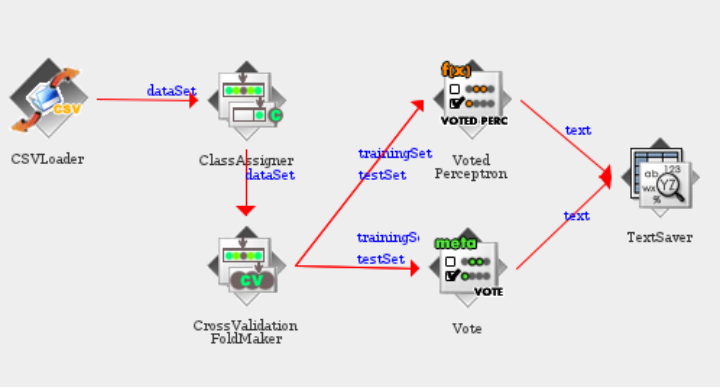
(a) Data cleaning workflow. A simplified version is shown here: not all filters are present.



(b) Data selection workflow.



(e) Classifier training workflow. Not all classifiers are included in this representation.



(f) Meta-learning workflow.

Figure 6.3: Weka KnowledgeFlow workflows.

to one or more types of attributes. We include also two techniques from the statistical field for comparison: regression analysis. Statistics are widely used within the biomedical field and therefore included here as a baseline.

We acknowledge the fact that there are many more methods available, however, we tried to create a "cross-section" of the many types of classifiers existing today. A clear focus is on the more widely used methods. We present each method together with a motivation why this particular method seems to be useful on our data as well as the tool/implementation used:

- Logistic regression [Hosmer Jr et al., 2013] — often used in statistical analysis, here used as comparison;
- Least squares [Borowiak, 2001] — often used in statistical analysis, here used as comparison;
- Bayesian network [Pearl, 2000] — often used in the medical field;
- Decision tree (C4.5) [Rokach, 2007] — baseline for many knowledge discovery methods;
- Random forest [Breiman, 2001] — baseline for many knowledge discovery methods;
- Neural network (multilayer perceptron with one hidden layer) [Rosenblatt, 1961] — especially useful for the large quantities of numerical data;
- Support vector machine (linear) [Cristianini and Shawe-Taylor, 2000] — baseline for many knowledge discovery methods;
- Subgroup discovery [Lavrač et al., 2004, Meeng and Knobbe, 2011] — identifying subgroups might be useful in describing possibly different forms of the disease.

For practical reasons mostly implementations from the Weka toolbox are used. We believe that the actual implementation of the algorithm is not affecting the results. Nor are we primarily interested in the actual classifying power, but more in the added value of the disjoint sets of attributes.

6.3.2 Quality metrics

A lot of different quality metrics have been used to describe the performance of classifiers. Most of them are defined in term of the confusion matrix containing the number of true positives, true negatives, false positives, and false negatives. Commonly used metrics include: precision and recall, sensitivity and specificity, and receiver operating characteristic. As we are trying to characterize the relative performance of several methods, we choose a method that can be expressed as a single number. We used the so-called weighted relative accuracy (WRAcc) [Todorovski et al., 2000]:

$$\text{WRAcc}(\text{Class} \leftarrow \text{Cond}) = p(\text{Cond}) \cdot (p(\text{Class}|\text{Cond}) - p(\text{Class})).$$

The WRAcc embodies a trade-off between standard accuracy and generality without sacrificing to much accuracy. Often this metric performs well and tends to yield fewer and simpler patterns, which are considered to be an asset in the biomedical field as usually the resulting patterns have to be explained in this field.

6.4 Experiments

In this section we will describe the computer experiments. Note that we are not primarily interested in the classification performance of the disease, but rather to investigate the effects of augmenting the separate sets with each other with regard to this classification problem.

In all experiments we use *stratified* 10-fold cross-validation. Each data set is randomly partitioned into ten equal size subsamples in such a way that the proportion of the cases versus the controls is constant. A single subsample is designated to be the validation data for the trained model, while the remaining nine subsamples are combined to train the model.

As a baseline we use the current standard risk prediction method derived from earlier studies, which is a non-linear function over all features captured within the risk factor set. The performance of this method on our three studies is described in Table 6.1.

Table 6.1: The performance of the current standard risk prediction method.

Study	WRAcc
Cohort Study I	79.08
Cohort Study II	84.90
Cohort Study III	87.13

6.4.1 Classification power of disjoint sets of attributes

As a first experiment we investigate the additional classification power gained by adding each disjoint data set. We did this for all three cohort studies. We took all combinations of the disjoint sets and calculated their respective predictive power, see Table 6.2. We do not show the combinations where the risk factors are included, because they result in every case in a WRAcc that is very close to the WRAcc of the risk factors alone. With the notable exception of the "all" data set, where all attributes (including the risk factors) are considered. Note that the set of FFAs is not available for Cohort Study II.

As can be observed from Table 6.2, the small set of risk factors is able to outperform all of the other combinations of (much larger) sets of attributes in term of absolute predictive power regardless of the classifier. However, some classifiers are able to outperform the risk factors at specific sets. For instance, neural networks seem to perform better on some of the sets of attributes that contain predominantly numerical data. Furthermore, in all cases providing all data to the classifier improves its prediction power. This is not generally true. As most classifiers use a heuristic method to combine certain attributes, it can be the case that adding data obscures the underlining patterns, which result in a reduced performance. This effect can also be observed in Table 6.2, e.g., when combining {NMR, SNP, FFA} which results sometimes in a lower WRAcc than the combination {NMR, FFA}.

Based on the results for the three cohort studies, we cannot draw statistically significant conclusions about the relative power of the groups of attributes. For this we should include additional cohort studies. The meta-analysis method

Table 6.2: The WRAcc for combinations of the disjoint sets of attributes.

Cohort Study I set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	80.98	73.98	59.96	74.63	78.07	58.09	66.59	62.03
risk factors	73.08	64.55	55.26	71.79	71.14	51.22	63.99	57.38
NMR	62.63	57.45	50.08	62.40	65.08	50.00	57.02	56.44
SNP	65.74	60.01	59.00	63.87	61.79	51.52	56.41	56.04
FFA	66.56	56.29	54.14	62.98	64.01	53.74	58.00	57.25
{NMR, SNP, FFA}	70.15	57.78	50.26	63.57	70.60	52.59	61.15	55.80
{NMR, SNP}	65.12	62.59	50.11	65.21	64.09	53.97	62.05	51.09
{NMR, FFA}	68.07	63.67	53.28	62.20	61.52	54.07	59.22	54.31
{SNP, FFA}	65.57	58.73	58.21	58.40	66.93	53.99	58.49	54.73
Cohort Study II set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	90.02	81.05	67.80	86.18	88.42	54.92	85.05	58.51
risk factors	89.98	76.50	74.61	85.98	83.60	52.76	83.73	68.59
NMR	75.61	61.57	56.47	65.99	70.52	57.63	67.72	55.76
SNP	69.61	67.13	60.66	72.61	72.98	57.01	77.53	61.25
{NMR, SNP}	72.70	60.01	57.51	72.93	72.65	59.13	63.17	51.81
Cohort Study III set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	93.97	79.32	77.60	84.72	87.35	70.88	83.85	77.35
risk factors	89.98	72.22	52.07	85.05	77.61	65.65	75.56	67.04
NMR	74.45	68.23	51.72	72.05	82.16	58.47	71.61	50.99
SNP	73.85	61.91	57.59	69.72	70.94	57.81	69.79	53.21
FFA	73.55	57.78	52.76	79.89	76.33	58.66	69.61	67.46
{NMR, SNP, FFA}	77.14	66.07	54.45	70.60	78.21	56.98	70.11	57.00
{NMR, SNP}	75.21	71.43	58.61	73.98	66.89	53.47	75.36	58.25
{NMR, FFA}	77.86	70.45	72.20	78.76	76.51	50.73	69.56	60.79
{SNP, FFA}	76.63	65.84	62.97	65.41	76.33	63.89	66.48	51.66

can then provide very useful insights for the classification problems of the cohort studies.

6.4.2 Using classifiers across cohort studies

In this second experiment, we are interested in the performance of the classifiers (trained and validated on their respective cohort studies) in Table 6.2 on other studies. We hope to acquire insight in the generality of the classifiers. In Table 6.3, we present the measured predictive power of the classifier trained and validated on Cohort Study I (as this is the largest) tested on Cohort Study II and Cohort Study III. Note that the set of FFAs is not available for Cohort Study II.

Table 6.3: The performance (WRAcc) of the classifiers trained on Cohort Study I and tested on Cohort Study II and Cohort Study III.

Cohort Study II set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	75.69	58.03	51.74	60.76	69.65	53.13	56.37	57.93
risk factors	61.08	61.34	52.49	63.13	64.85	49.70	67.97	56.05
NMR	59.90	51.00	50.93	62.85	67.60	52.88	56.62	49.10
SNP	58.35	66.65	59.59	59.47	60.93	53.54	53.12	45.64
{NMR, SNP}	56.67	54.11	36.01	63.43	72.22	48.28	68.04	52.32
Cohort Study III set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	54.01	49.98	36.72	53.84	57.42	64.42	48.44	43.47
risk factors	57.77	42.50	28.42	52.18	67.53	55.16	61.57	49.83
NMR	55.31	50.87	32.19	59.29	64.09	46.22	61.01	53.77
SNP	41.52	46.40	42.65	50.71	61.71	59.21	58.38	38.71
FFA	46.46	47.73	53.07	52.13	63.96	59.29	55.50	40.19
{NMR, SNP, FFA}	51.80	55.18	47.43	50.43	47.54	58.22	52.49	44.19
{NMR, SNP}	52.83	50.86	37.63	56.65	69.83	55.98	50.20	34.31
{NMR, FFA}	48.39	39.44	30.73	49.68	65.93	58.51	50.64	49.15
{SNP, FFA}	58.05	44.58	29.26	59.19	55.67	58.19	58.49	50.07

The results in Table 6.3 are not as good as expected. Although the performance on Cohort Study I is reasonable good, the performance on Cohort

Study III is not very good, as, most scores are just above 50%. Apparently the model for Cohort Study I is not capable of predicting the disease in more general cases. The predictive power maintained on Cohort Study I is notably better. A possible explanation being the low number of cases in Cohort Study III. It might even be possible that these cases are of a different type with regard to the predominant cases characterized in Cohort Study I as overfitting seems not to be the problem when training on the Cohort Study I.

The remaining forms of cross model testing are not shown here as they yield significantly less performance results. Again presumably because of the low ratio of cases versus controls in this study.

6.4.3 Combining all data from different studies

As is apparent from the experiments in Section 6.4.2, the generality of the constructed classifiers can be improved. We introduce a new experiment: instead of training and validating on a separate study we will combine all data into a single data set. 90% of this data set is used to train and validate the classifiers (as usual using 10-fold cross validation). The resulting classifiers are then tested on the respective 10% of the disjoint sets from their original cohort studies. Thus avoiding the pitfall of overfitting. These results are shown in Table 6.4.

The predictive power described in Table 6.4 is indeed more promising than in Table 6.3. However, we lose a lot of accuracy on all of the separate studies at the benefit of a more stable (general) classifier. We feel that the added benefit of stability does not outweigh the medical consequences of losing that much performance, and is, therefore, not satisfactory in practice.

6.4.4 Hierarchical approach

Next we present a hierarchical approach to improve the overall performance of the classifiers on the disjoint sets of attributes. Based on the observation in Table 6.2 that some classifiers yield better results on certain sets. We want to make use of this feature by applying the best performing classifier on each of the disjoint sets, and combine their predictions in some way. Note that we will

Table 6.4: WRAcc of the classifiers trained on all data combined and tested on disjoint sets of attributes.

Cohort Study I set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	58.23	59.19	51.70	62.63	65.30	69.52	58.04	46.51
risk factors	54.40	58.03	45.04	57.28	57.56	57.10	55.21	40.09
NMR	52.12	42.76	45.95	57.91	59.58	49.18	52.54	40.94
SNP	44.50	43.31	50.86	53.83	50.53	62.42	56.36	36.24
{NMR, SNP}	52.87	40.10	49.21	58.85	48.43	54.47	39.50	43.53

Cohort Study II set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	78.60	62.16	62.75	69.94	72.45	59.24	70.33	65.39
risk factors	73.15	53.76	45.12	57.57	60.89	56.31	50.48	58.42
NMR	64.36	59.89	59.15	62.02	67.19	52.38	65.13	55.43
SNP	61.82	62.74	54.49	60.99	66.53	50.89	50.69	56.50
{NMR, SNP}	68.79	54.79	62.04	65.00	64.20	48.63	69.18	50.98

Cohort Study III sets(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	65.20	59.21	64.09	66.01	70.45	58.88	65.83	62.41
risk factors	59.50	59.41	62.51	55.83	69.86	58.14	54.11	47.66
NMR	60.81	52.77	50.66	53.76	68.82	49.10	56.10	58.86
SNP	55.78	51.16	45.95	63.36	66.35	54.66	45.71	45.90
{NMR, SNP}	60.16	51.57	61.45	59.84	55.08	52.61	65.55	59.26

consider the single sets only. Applying this method to all combinations of the disjoint sets will be regarded as future work. Reminding the results from the experiments performed in Section 6.4.2 and Section 6.4.1, we will once again consider the studies separately.

We will use two methods of combining the predictions of the individual classifiers: majority voting and a linear perceptron, a simple form of a neural network consisting of only an input layer (the predictions) and an output layer

of one node: the ultimate prediction based on the predictions of the individual classifiers. In case of majority voting we do not need to train the hierarchical method, so no addition data is required. This is not the true for in case of the linear perceptron that needs to be trained as well. We use the same partitioning strategy as used in Section 6.4.3 in order to avoid overfitting. In Table 6.5 we present the performance of the hierarchically linked classifiers trained on their respective studies, but selected based on their individual performance on Cohort Study III, and, in case of the classifiers trained on the Cohort Study III data, the data from Cohort Study I.

Table 6.5: The WRAcc of the hierarchically linked classifiers.

study	majority	linear perceptron
Cohort Study I	73.43	82.83
Cohort Study II	83.28	90.74
Cohort Study III	92.33	94.55

As is apparent in Table 6.5, we are able to improve upon the predictive power of the individual classifiers as is shown in Table 6.2 when using a linear perceptron. The majority voting approach seems to be inferior to using the flat data. Probably, this is caused by the fact that all data sets are weighted equally, where as analyzing the data directly retains the possibility of weighting groups and individual attributes differently. The (small) increase in performance of the hierarchical approach by using a linear perceptron might be explained by a divide-and-conquer argument. By presenting the classifiers with smaller data set they are more easily capable of fitting a model there on. Apparently, these individual models can be combined just by using a simple weighting scheme.

6.5 Conclusions

In this chapter we investigated the added benefits of augmenting large cohort studies with disjoint sets of attributes and data from other studies. We provided

a “cross-sectional” study of different classifiers on the heterogeneous data available within these studies. As expected, knowledge discovery on this kind of studies is not a trivial task. We compared our results with a baseline standard risk prediction method used in literature. We shown that our methods are able to match its performance, and, in some cases, are able to outperform it. In particular an hierarchical approach seem to yield good results. We have demonstrated that applying machine learning techniques combined with workflow tooling are valuable in solving this task.

As this is rather a small preliminary study, extending this research onto more large cohort studies, and investigating for different phenotypes is regarded to be a valid pointer towards future research.

Chapter 7

Conclusions and Future Work

In this chapter we summarize the findings of the previous chapters in relation to the research questions presented in Chapter 1. Furthermore, we present some directions for future work.

In Chapter 3 we asked the research question: “Can we automatically generate concise, meaningful HGVS descriptions from DNA sequences in linear time?”. We have designed and implemented an algorithm to extract HGVS descriptions from raw DNA sequences and we made this algorithm efficient for highly similar sequences, i.e., expected linear run-time for whole chromosomes. One of the focal points is the length of the resulting descriptions. Using a weighting scheme our algorithm tries to minimize these. In general, we cannot guarantee that a generated description is minimal, however, we have proven the usefulness of our descriptions by the automatic curation of databases. In addition, we proposed an extension of the HGVS nomenclature for the description of so-called transpositions.

We shifted our focus to protein sequences in Chapter 4, were we asked the research question: “Can we automatically generate HGVS descriptions from protein sequences as well?”. Based on the algorithm from Chapter 3, we developed a method that can automatically generate protein sequence descriptions following the HGVS nomenclature. A particular emphasis lies on the detection and annotation of so-called frameshift variants. One of the most important results from this chapter is that our method is able to effectively detect and annotate these frameshift variants from the raw protein sequences without having access to the underlying DNA sequences. This is a prerequisite first step towards unambiguous protein variant descriptions. In addition to the detection of frameshift variants, we can also accurately calculate their corresponding probabilities. The experiments show that, in particular within a species, large frameshift variants commonly exist, which may be an indication of their evolutional significance.

Chapter 5 deals with the research question: “Can we efficiently generate descriptions for regions with repeated sequences?”. In this chapter we present a method, again based on the algorithms from Chapter 3, that automatically generates reference based descriptions for regions with short tandem repeats. We presented an Algorithm to automatically detect the repeat units present in a sequence. These repeat units can then be used to generate a description. Alternatively, annotated repeat units from literature can be used. In addition to the description of the repeat structure, we have an integrated approach

that also describes the flanking regions relative to this repeat structure. We advocated the use of LRG reference sequences for generating descriptions for short tandem repeats.

Finally, in Chapter 6, we moved away from the generation of descriptions and we explored disjoint sets of attributes in large cohort studies. We asked the research questions: "Can we say something about the relative power of the (combinations of) sets of attributes?" and "Can data from different cohort studies be used to augment classifying power for a single study?". We provided a "cross-sectional" study of different classifiers on the heterogeneous data available within these cohort studies. As expected, knowledge discovery on this kind of studies is not a trivial task. We compared our results with a baseline standard risk prediction method used in literature. We shown that our methods are able to match its performance, and, in some cases, are able to outperform it.

7.1 Future Work

In Chapters 3, 4 and 5 we propose extensions to the HGVS nomenclature. Currently none of these proposed extensions are part of the nomenclature. Some effort will be required to either make our propositions part of the HGVS nomenclature or to adapt our algorithms and methods to the nomenclature. In any case, in the current situation the nomenclature and the tooling to not match completely. In order to effectively develop and reason about the nomenclature we foresee that a formal specification of the nomenclature is needed.

Many features of the HGVS nomenclature are unsupported in our implementations. In particular, there is no support for nested variants. Whether nested variants are appropriate in descriptions remains an open question. As yet, we do not support transpositions involving more than one chromosome. As the HGVS nomenclature committee is working on new guidelines involving junctions of more than one chromosome, support will become necessary in the foreseeable future. As sequencing technology is evolving quickly, other topic will gain importance and momentum, e.g., graph based reference genomes.

Our work in Chapter 5 needs to be evaluated in a forensic setting before the

proposed extensions can be implemented in the HGVS nomenclature. This also means a strategy for reference sequences for STR alleles and their respective repeat region annotation and the repeat unit set standardization per locus.

Finally, with regard to the explorative study in Chapter 6, the observations and results should be validated; extending this research onto more large cohort studies, and investigating for different phenotypes is regarded to be a valid pointer towards future research.

Bibliography

[Aartsma-Rus et al., 2006] Aartsma-Rus, A., Van Deutekom, J. C. T., Fokkema, I. F., Van Ommen, G.-J. B., and Den Dunnen, J. T. (2006). Entries in the leiden duchenne muscular dystrophy mutation database: An overview of mutation types and paradoxical cases that confirm the reading-frame rule. *Muscle & nerve*, 34(2):135–144.

[Abbasi, 1997] Abbasi, S. (1997). Longest common consecutive substring in two random strings. Technical report, Center for Discrete Mathematics & Theoretical Computer Science.

[Alberts et al., 1995] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., Watson, J. D., and Grimstone, A. V. (1995). Molecular biology of the cell (3rd edn). *Trends in Biochemical Sciences*, 20(5):210–210.

[Anvar et al., 2014] Anvar, S. Y., van der Gaag, K. J., van der Heijden, J. W. F., Veltrop, M. H. A. M., Vossen, R. H. A. M., de Leeuw, R. H., Breukel, C., Buermans, H. P. J., Verbeek, J. S., de Knijff, P., et al. (2014). TSSV: a tool for characterization of complex allelic variants in pure and mixed genomes. *Bioinformatics*, 30(12):1651–1659.

[Beaudet and Tsui, 1993] Beaudet, A. L. and Tsui, L.-C. (1993). A suggested nomenclature for designating mutations. *Human Mutation*, 2(4):245–248.

[Beutler, 1993] Beutler, E. (1993). The designation of mutations. *American journal of human genetics*, 53(3):783.

[Borowiak, 2001] Borowiak, D. (2001). Linear Models, Least Squares and Alternatives. *Technometrics*, 43(1):99–99.

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

[Breiman, 2001] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

[Compeau et al., 2011] Compeau, P. E. C., Pevzner, P. A., and Tesler, G. (2011). How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991.

[Crick, 1958] Crick, F. H. C. (1958). On protein synthesis. In *Symposia of the Society for Experimental Biology*, volume 12, page 8.

[Crick et al., 1970] Crick, F. H. C. et al. (1970). Central dogma of molecular biology. *Nature*, 227(5258):561–563.

[Cristianini and Shawe-Taylor, 2000] Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

[Delaye et al., 2008] Delaye, L., DeLuna, A., Lazcano, A., and Becerra, A. (2008). The origin of a novel gene through overprinting in escherichia coli. *BMC Evolutionary Biology*, 8(1):31–41.

[den Dunnen et al., 2000] den Dunnen, J. T., Antonarakis, S. E., et al. (2000). Mutation nomenclature extensions and suggestions to describe complex mutations: A discussion. *Human Mutation*, 15(1):7–12.

[den Dunnen et al., 2016] den Dunnen, J. T., Dalgleish, R., Maglott, D. R., Hart, R. K., Greenblatt, M. S., McGowan-Jordan, J., Roux, A.-F., Smith, T., Antonarakis, S. E., and Taschner, P. E. M. (2016). Hgvs recommendations for the description of sequence variants: 2016 update. *Human mutation*.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Gettings et al., 2015] Gettings, K. B., Aponte, R. A., Vallone, P. M., and Butler, J. M. (2015). STR allele sequence variation: current knowledge and future issues. *Forensic Science International: Genetics*, 18:118–130.

[Gîrdea et al., 2010] Gîrdea, M., Noé, L., and Kucherov, G. (2010). Back-translation for discovering distant protein homologies in the presence of frameshift mutations. *Algorithms for Molecular Biology*, 5(6).

[Golomb, 1966] Golomb, S. (1966). Run-length encodings (corresp.). *IEEE Transactions on Information Theory*, 12(3):399–401.

[Gusfield, 1997] Gusfield, D. (1997). *Algorithms on strings, trees and sequences: Computer science and computational biology*. Cambridge University Press.

[Hall et al., 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

[Hoogenboom et al., 2016] Hoogenboom, J., van der Gaag, K. J., de Leeuw, R. H., Sijen, T., de Knijff, P., and Laros, J. F. J. (2016). FDSTools: A software package for analysis of massively parallel sequencing data with the ability to recognise and correct STR stutter and other PCR or sequencing noise. *in preparation*.

[Hosmer Jr et al., 2013] Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied Logistic Regression*. Wiley.

[Laros et al., 2011] Laros, J. F. J., Blavier, A., den Dunnen, J. T., and Taschner, P. E. M. (2011). A formalized description of the standard human variant nomenclature in extended backus-naur form. *BMC bioinformatics*, 12(Suppl 4):S5.

[Lavrač et al., 2004] Lavrač, N., Kavšek, B., Flach, P., and Todorovski, L. (2004). Subgroup discovery with CN2-SD. *The Journal of Machine Learning Research*, 5:153–188.

[MacArthur et al., 2014] MacArthur, J. A. L., Morales, J., Tully, R. E., Astashyn, A., Gil, L., Bruford, E. A., Larsson, P., Flicek, P., Dalgleish, R., Maglott,

D. R., et al. (2014). Locus Reference Genomic: reference sequences for the reporting of clinically relevant sequence variants. *Nucleic acids research*, 42:873–878.

[Meeng and Knobbe, 2011] Meeng, M. and Knobbe, A. (2011). Flexible Enrichment with Cortana Software Demo. In *Proceedings of BeneLearn*, pages 117–119.

[Meselson and Stahl, 1958] Meselson, M. and Stahl, F. W. (1958). The replication of dna in escherichia coli. *Proceedings of the National Academy of Sciences*, 44(7):671–682.

[Nakamura et al., 2000] Nakamura, Y., Gojobori, T., and Ikemura, T. (2000). Codon usage tabulated from international dna sequence databases: status for the year 2000. *Nucleic acids research*, 28:292–292.

[Oinn et al., 2004] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A., and Li, P. (2004). Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 20(17):3045–3054.

[Pearl, 2000] Pearl, J. (2000). *Causality: Models, Reasoning and Inference*, volume 29. Cambridge University Press.

[Robinson and Thain, 2013] Robinson, C. and Thain, D. (2013). Automated packaging of bioinformatics workflows for portability and durability using makeflow. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, WORKS '13, pages 98–105, New York, NY, USA. ACM.

[Robinson et al., 2014] Robinson, J., Halliwell, J. A., Hayhurst, J. D., Flicek, P., Parham, P., and Marsh, S. G. E. (2014). The IPD and IMGT/HLA database: Allele variant databases. *Nucleic Acids Research*, pages 423–431.

[Rokach, 2007] Rokach, L. (2007). *Data Mining with Decision Trees: Theory and Applications*, volume 69. World Scientific.

[Rosenblatt, 1961] Rosenblatt, F. (1961). Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms. Technical report, DTIC Document.

[Sheetlin et al., 2014] Sheetlin, S. L., Park, Y., Frith, M. C., and Spouge, J. L. (2014). Frameshift alignment: statistics and post-genomic applications. *Bioinformatics*, 30(24):3575–3582.

[Smith and Waterman, 1981] Smith, T. F. and Waterman, M. S. (1981). Identification of Common Molecular Subsequences. *Journal of molecular biology*, 147(1):195–197.

[Taschner and den Dunnen, 2011] Taschner, P. E. M. and den Dunnen, J. T. (2011). Describing structural changes by extending HGVS sequence variation nomenclature. *Human Mutation*, 32(5):507–511.

[Tichy, 1984] Tichy, W. F. (1984). The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems (TOCS)*, 2(4):309–321.

[Todorovski et al., 2000] Todorovski, L., Flach, P., and Lavrač, N. (2000). *Predictive Performance of Weighted Relative Accuracy*. Springer.

[UniProt Consortium, 2015] UniProt Consortium (2015). UniProt: a hub for protein information. *Nucleic acids research*, 43:204–212.

[Vis and Kok, 2014] Vis, J. K. and Kok, J. N. (2014). *Meta-analysis of Disjoint Sets of Attributes in Large Cohort Studies*, pages 407–419. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Vis et al., 2015] Vis, J. K., Vermaat, M., Taschner, P. E. M., Kok, J. N., and Laros, J. F. J. (2015). An Efficient Algorithm for the Extraction of HGVS Variant Descriptions from Sequence. *Bioinformatics*, 31(23):3751–3757.

[Wagner and Fischer, 1974] Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173.

[Wagner and Lowrance, 1975] Wagner, R. A. and Lowrance, R. (1975). An extension of the string-to-string correction problem. *Journal of the ACM (JACM)*, 22(2):177–183.

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

[Weischenfeldt et al., 2013] Weischenfeldt, J., Symmons, O., Spitz, F., and Korbel, J. O. (2013). Phenotypic impact of genomic structural variation: insights from and for human disease. *Nature Reviews Genetics*, 14(2):125–138.

[Wildeman et al., 2008] Wildeman, M., van Ophuizen, E., den Dunnen, J. T., and Taschner, P. E. M. (2008). Improving sequence variant descriptions in mutation databases and literature using the Mutalyzer sequence variation nomenclature checker. *Human Mutation*, 29(1):6–13.

[Wolstencroft et al., 2013] Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., et al. (2013). The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic acids research*, 41(W1):W557–W561.

Samenvatting

In dit proefschrift wordt een onderdeel van de bio-informatica bestudeerd. Er is bijzondere aandacht voor het berekenen van beschrijvingen van moleculaire sequenties. Hierbij wordt een (geobserveerde) sequentie beschreven ten opzichte van een referentiesequentie. Moleculaire sequentie komen veel voor in de biologie. Hieronder wordt onder andere verstaan: DNA (en RNA) sequenties en eiwitsequenties. In een klinische omgeving worden veelal de individuele verschillen van deze sequentie bestudeerd en in verband gebracht met allerlei fenotypen (onder andere aandoeningen). Tijdens dit proces spelen de beschrijvingen een belangrijke rol. Veelal worden deze beschrijvingen gegeven in een domein specifieke taal; de HGVS nomenclatuur. Deze schrijft voor hoe bepaalde typen varianten, zoals substituties, inserties en deleties, dienen te worden omschreven. Het streven hier is een ondubbelzinnig uitlegbare beschrijving voor alle varianten in een bepaald individu. Deze HGVS beschrijvingen worden al geruime tijd intensief gebruikt, maar een effectieve methode voor het berekenen (construeren) van deze beschrijvingen ontbrak.

In Hoofdstuk 3 wordt een effectieve methode beschreven voor het genereren van dit soort beschrijvingen. Daarin zijn kwaliteiten zoals de lengte van de berekende beschrijving, de complexiteit van de berekening, maar ook de biologische betekenis van de beschrijving van belang. Zo wordt er onder meer rekening gehouden met het complementaire karakter van DNA. In dit hoofdstuk wordt ook een aanvulling op de HGVS nomenclatuur voorgesteld; (samengestelde) transposities, waarin delen van de referentiesequentie kunnen worden verplaatst en/of gekopieerd. Wij laten zien dat onze methode in staat is beschrijvingen te genereren voor complete chromosomen en dat de door

onze methode gegenereerde beschrijvingen nuttig kunnen worden ingezet voor het cureren van databases.

In Hoofdstuk 4 verleggen wij de aandacht van DNA naar eiwitsequenties. Eiwitten worden in tripletten gecodeerd in bepaalde delen van het DNA. Juist deze tripletcodering zorgt ervoor dat veranderingen in het DNA leiden tot zogenaamde verschuivingen in de eiwitsequentie, omdat de tripletten op een andere manier over de DNA-sequentie zijn verdeeld. Deze verschuivingen geven aanleiding tot het maken van beschrijvingen op eiwitsequenties. In dit hoofdstuk gebruiken wij deels technieken uit Hoofdstuk 3 en deels nieuwe technieken voor het genereren van beschrijvingen voor eiwitsequenties. In het bijzonder valt te berekenen van de waarschijnlijkheid is wanneer een kandidaatverschuiving wordt beschouwd. Vervolgens tonen wij aan dat deze verschuivingen, in het bijzonder binnen een soort, veelvuldig voorkomen.

In Hoofdstuk 5 kijken wij naar een bijzondere vorm van DNA-sequenties; sequenties waarin vele kleine herhalingen voorkomen. Deze herhalingen hebben een bijzonder gebruik in forensisch onderzoek. Op de eerste plaats geven wij een methode voor het vinden van kandidaatherhalingen in een sequentie. Ten tweede presenteren wij een methode die, gebruikmakend van een verzameling van herhalingen, een beschrijving van een herhalingsstructuur geeft. En ten slotte, kan deze beschrijven worden gecombineerd met een beschrijving van de omliggende sequenties ten opzichte van de herhalingsstructuur. Naast de methoden voor het maken van deze beschrijvingen, geven wij ook aanwijzingen voor het correct gebruik van referentiesequenties in de forensische toepassing.

Wij nemen afstand van het berekenen van beschrijvingen in Hoofdstuk 6. In dit hoofdstuk kijken wij op een exploratieve manier naar het datalandschap waarin de bio-informatica veelal opereert. De analyse van grote cohortstudies, waarin groepen heterogene data kunnen worden samengebracht, staat centraal. Het effect van het samenvoegen van verschillende groepen data wordt onderzocht in het kader van het kunnen classificeren van een fenotype uit deze data. Hiertoe worden enkele veelgebruikte classificatietechnieken toegepast. Daarnaast wordt ook gekeken naar het samenvoegen van data uit andere (ongerelateerde) cohortstudies, wederom met als doel het classificeren van een fenotype. De resultaten worden steeds vergeleken met klassieke risicofactoren

voor dat fenotype. Het blijkt dat deze risicofactoren veelal goede classificatiekriteria zijn en dat naïef het toevoegen van groepen data niet direct tot een beter resultaat leidt. Slecht wanneer alle beschikbare data wordt gecombineerd in een hiërarchische manier wordt een marginaal beter resultaat behaald.

Curriculum Vitae

Jonathan Vis is geboren op 25 juli 1983 te Leuven (België). Hij behaalde in 2001 zijn VWO-diploma aan het Johan de Witt gymnasium te Dordrecht. Vervolgens voltooide hij de studie Werktuigbouwkunde aan de Hogeschool Rotterdam & Omstreken in 2006. In 2009 behaalde hij zijn bachelor Informatica aan de Universiteit Leiden, gevolgd door de master Computer Science eveneens te Leiden. Naast zijn studie voltooide hij in 2000 de vakopleiding Edelsteenkunde van het Nederlandsch Genootschap voor Edelsteenkunde te Gouda en was hij actief lid van het Genootschap en later bestuurslid van het Gemmologisch Gilde Nederland te Utrecht.

Jonathan was ook betrokken als assistent bij de vakken Computerarchitectuur, Programmeermethoden, Algoritmiek en Kunstmatige Intelligentie allen vakken van de opleiding Informatica te Leiden. Van 2011 tot 2015 deed hij promotieonderzoek aan de Universiteit Leiden onder leiding van prof. dr. Eline Slagboom, prof. dr. Joost Kok en dr. Jeroen Laros. De resultaten van dat onderzoek zijn samengebracht in dit proefschrift.

Tijdens zijn promotieonderzoek heeft Jonathan ook onderzoek gedaan naar de complexiteit van spellen hetgeen in een aantal publicaties heeft geresulteerd.

Dankwoord

Ik bedank mijn collegae van het LIACS waaronder de leden van de “koffieclub”: Frans Birrer, Hendrik Jan Hoogeboom, Jan van Rijn, Marijn Schraagen, Frank Takes en mijn kamergenoot Jurriaan Rot voor de onderhoudende en interessante discussies. Mijn bijzondere dank gaat uit naar Walter Kosters die mij heeft overtuigd aan het promotietraject te beginnen.

Jonathan Vis

Publication List

van Rijn, J.N., Takes, F.W. & Vis, J.K. (2015), The Complexity of Rummikub Problems. In: *Proceedings of the 27th Benelux Conference on Artificial Intelligence*

Vis J.K., Vermaat M., Taschner P.E.M., Kok J.N. & Laros J.F.J. (2015), An efficient algorithm for the extraction of HGVS variant descriptions from sequences, *Bioinformatics* 31(23): 3751–3757.

Hoogeboom H.J., Kosters W.A., Rijn J.N. van & Vis J.K. (2014), Acyclic Constraint Logic and Games, *ICGA Journal* 37(1): 3–16.

van Rijn, J.N., Vis & J.K. (2014), Endgame Analysis of Dou Shou Qi, *ICGA Journal* 38: 120–124.

Vis J.K. & Kok J.N. (2014), Meta-analysis of Disjoint Sets of Attributes in Large Cohort Studies. In: *Proceedings ISoLA 2014* no. LNCS 8803: Springer. 407–419.

van Rijn J.N. & Vis J.K. (2013), Complexity and Retrograde Analysis of the Game Dou Shou Qi. In: *Proceedings of the 25th Benelux Conference on Artificial Intelligence*. 239–246.

Vis J.K., Kosters W.A. & Batenburg K.J. (2011), Discrete Tomography: A Neural Network Approach. In: *Proceedings of 23rd Benelux Conference on Artificial Intelligence*. 328–335.

Vis J.K., Kosters W.A. & Terroba A. (2010), Tennis Patterns: Player, Match and Beyond. In: *Proceedings of 22nd Benelux Conference on Artificial Intelligence*.

Terroba A., Kosters W.A. & Vis J.K. (2010), Tactical Analysis Modeling through Data Mining: Pattern Discovery in Racket Sports. In: *International Conference on Knowledge Discovery and Information Retrieval*.