Computational challenges in genome wide association studies: data processing, variant annotation and epistasis

Pablo Cingolani

PhD.

School of Computer Science - Bioinformatics

McGill University

Montreal,Quebec

March 2015

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Doctor of Philosophy

ABSTRACT

Abstract...abstract...abstract...abstract...abstract...abstract...abstract...abstract...abstract

ABRÉGÉ

Abstract...abstract...abstract...abstract...abstract...abstract...abstract...abstract...abstract

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Motivation

How does your DNA influence your risk of getting a disease? Contrary to popular belief, your future health is not "hard wired" in your DNA. Only in a few diseases, referred as "Mendelian diseases", there are well known, almost certain, links between genetic mutations and disease susceptibility. For the majority of what are known as "complex traits", such as cancer or diabetes, genomic predisposition is subtle and, so far, not fully understood.

With the rapid decrease in the cost of DNA sequencing, the complete genome sequence of large cohorts of individuals can now be routinely obtained. This wealth of sequencing information is expected allow the identification of genetic variations linked to complex traits. In this work, I investigate the analysis of genomic data in relation to complex diseases, which offers a number of important computational and statistical challenges. We tackle several steps necessary for the analysis of sequencing data and identifying the links to disease. Each step, which will correspond to a chapter in my thesis, is characterized by very different problems that need to be addressed.

i The first step is to analyze large amounts of information generated by sequencers to obtain a set of "genomic variants" that distinguish each individual. To address these big data processing problems, Chapter 2 shows how we designed a programming language (BigDataScript or BDS), that simplifies the creation robust, scalable data pipelines.

ii Once genomic variants are obtained, we need to prioritize and filter them to discern which variants should be considered "important" and which ones are likely to be less relevant. In this process, known as "functional variant annotation" or simply "variant annotation", we calculate how the protein product would be affected and add information from relevant genomic databases (such as protein structure, deleteriousness scores or how often the variant is present in a population). We created SnpEff & SnpSift [**?**, **?**] packages that, using optimized algorithms, solve several annotation problems: a) standardize the annotation process, b) calculate putative genetic effects, c) estimate genetic impact, d) add several sources of genetic information, and e) facilitate variants filtering. We applied our methods in two large Genome Wide Association Studies (GWAS) for type II diabetes projects, in order to prioritize variants for statistical analysis. As a result of these studies, novel genes associated with diabetes and glycemic traits were found.

iii Finally, we address the problem of finding associations between "interacting genetic loci" and disease. One of the main problems in GWAS, known as "missing heritability", is that most of the phenotypic variance attributed to genetic causes remains unexplained. Since interacting genetic loci have been pointed out as one of the possible causes of missing heritability, finding links between such interactions and disease has great significance in the field. We propose a methodology to increase the statistical power of this type of approaches by combining population-level genetic information with evolutionary information.

In the rest of this introduction we give the background required to understand the material shown in Chapters 2 to 5 while providing motivations for our research. The transformation of raw sequencing data into biological insight in the aetiology of complex disease poses a series of computational, analytical, algorithmic and methodological challenges that we address in the rest of this thesis.

1.1.1 Reference genome and genetic variants

DNA is composed of four basic building blocks, called “bases” or “nucleotides”. These four nucleotides, usually abbreviated $\{A, C, G, T\}$, are Adenine, Cytosine, Guanine, and Thymine. Bases form pairs, either as $A - T$ or $C - G$, that pile-up forming two long polymers, with backbones that run in opposite directions giving rise to a double-helix structure. Arbitrarily, one of the polymers is called the positive strand and the other is called the negative strand.

The human genome has a total of 3 Giga-base-pairs (Gb), and those bases are divided into 23 chromosomes. We have two copies of each “autosomal” chromosomes, one inherited from our mother and one from our father. There are 22 autosomal chromosomes. The longest, being roughly 250 Mega-bases (Mb), is called “Chromosome 1” and the shortest, being 50 Mb is called “chromosome 22”. We also have two sex chromosomes, called ’X’ and ’Y’.

In order to be able to compare different objects length, we need some reference measure, such as the reference meter. Similarly, in order to compare DNA from different individuals (or samples), we need a “reference genome”. The human reference genome (e.g. GRCh37) does not correspond to the DNA of any particular person, but to a “mosaic” of thirteen anonymous volunteers from Buffalo, New York [**?**].

When samples are sequenced, the DNA is compared to the “reference genome”. Most of the DNA is the same, but there are differences. These differences, generically known as “genomic variants” (or “variants”, for short), describe the particular genetic makeup of each individual. There are several different ways a sample can differ from a reference genome. These are known as “variant types” and can be roughly categorized in the following way:

Single nucleotide variants (SNV) or Single nucleotide polymorphism (SNP) are the simplest and more common variants produced by single base difference (e.g. a base in the reference genome, at a given coordinate, is an A, whereas the sample is C). There are several biological mechanisms responsible for this type of variants: i) replication errors, ii) errors introduced by DNA repair mechanism, iii) deamination (a base is changed by hydrolysis which may not be corrected by DNA repair mechanisms), iv) tautomerism (and alteration on the hydrogen bond that results in an incorrect pairing).

Multiple nucleotide polymorphism (MNP) are differences of more than one base (e.g. reference is ACG whereas the sample is TGC).

Insertions (INS) refer to a sample having extra base(s) compared to the reference genome (e.g. reference is AT and sample is ACT). Some small insertion are usually attributed to DNA polymerase slipping and replicating the same base/s (this produces a type of insertion known as duplication). Large insertions are can be caused by unequal cross-over event (during meiosis) or transposable elements.

Deletions (DEL) are the opposite of insertions, the sample has some base(s) removed respect to the reference genome (e.g. reference is ACT and sample is AT). As in the case of insertions, deletions can also be caused by ribosomal slippage, cross-over events during meiosis and transposable elements.

Mixed variants can happen as a more complex combinations of combining SNV/MNP + Ins/Del.

Copy number variations (CNVs) arise when the sample has two or more copies of the same genomic region (e.g. a whole gene that has been duplicated or triplicated) or conversely, when the sample has less copies than the reference genome. Copy number variations can be attributed to problem during homologous recombination events.

Rearrangements are some complex variants that involve joining different regions (e.g. a translocation between chromosomes). Inversions, a type of rearrangement, result from a whole genomic region being inverted. These types of mutations are often attributed to cross-over events during meiosis.

As humans have two copies of each chromosome, variants could affect zero, one or two of the chromosomes and are called "homozygous reference", "heterozygous", and "homozygous alternative" respectively. Variants are also be classified on how common they are within the population: common, low frequency, or rare (see sections **??**). How these types of genetic variants influence traits or risk of disease is a topic of intense research that will be discussed throughout this thesis.

Proteins are composed by chains of amino acids and, as explained by the central dogma of biology, DNA is the template that instructs cellular machinery how to produce proteins. There are 4 bases in the DNA. There are 20 amino acids, which are the building blocks of all proteins. Each of the twenty amino acids is encoded by a group of three DNA bases called "codon". More than one codon can code for the same amino acid (i.e. $4^3 = 64$ codons > 20 amino acids) allowing for code redundancy. Additionally, there are codons that mark the end of the protein, these are called "STOP" codons and signal molecular machinery to end the transcription process. So variations in DNA may sometimes have direct effects on the protein product. We will talk about this in section ?? and Chapter 3 where we cover the topic of "functional annotations".

1.1.2 DNA and disease

It would be fair to say that the Garrod family was fascinated by urine. As a physician at Kings College, Alfred Baring Garrod, discovered gout related abnormalities in uric acid [?]. His son, Sir Archibald Garrod, was interested in a condition known as alkaptonuria, in which children are mostly asymptomatic except for producing brown or black urine, but by the age of 30 individuals develop pain in joints of the spine, hips and knees. In 1902, Archibald observed that the family inheritance pattern of alkaptonuria resembled Mendels recessive pattern and postulated that a mutation in a metabolic gene was responsible for the disease. Publishing his finding he gave birth to a new field of study known as "Human biochemical genetics" [?].

Diseases having simple inheritance patterns, such as Cystic fibrosis, Phenylketonuria and Huntington's are also known as Mendelian diseases [?]. The genetic components of several Mendelian diseases have been discovered since the mechanism was first elucidated by Garrod in 1902 and the process has been accelerated in recent years, thanks to the application of DNA sequencing techniques [?].

In complex diseases (or complex traits), such as diabetes, cancer or Alzheimers, affected individuals cannot be segregated within pedigrees (i.e. no patterns of inheritance can be identified). As opposed to Mendelian diseases the aetiologically of complex traits is complicated due to factors such as: incomplete penetrance (symptoms are not always present in individuals who have the disease-causing mutation), oligogenic inheritance (characterized by more than one gene) and genetic heterogeneity (caused by any of a large number of alleles). This makes it difficult to pinpoint the genetic variants that increase risk of complex disease.

1.1.3 Type II diabetes

Although this thesis focusses on the development of computational approaches that could be applied to the study of a number of complex diseases, our focus has been on type II diabetes mellitus (T2D), a complex disease first described by the Egyptians in 1500 BCE. Later the Greeks in 230 BCE used the term "diabetes" meaning "pass through" (or "siphon") denoting the constant thirst and frequent urination of the patients. In the 1700s the term "mellitus" (from honey) was added to denote that the urine was sweet and would "attracts ants".

Diabetes symptoms include frequent urination, thirst, and constant hunger, high blood sugar (hyperglycemia) and insulin resistance. Long term complication from T2D may include eyesight problems, heart disease, strokes and kidney failure. Type II diabetes, is highly correlated with obesity and disease rate has increased dramatically during the last 50 years. According to the World Health Organisation the prevalence of diabetes is 9% in adults and an estimated 1.5 millions deaths were caused by diabetes in 2012 [?], which is predicted to be the 7th leading cause of death by 2030. The costs associated to treating diabetes patients only in the U.S. are estimated around $245 billion dollars.

In recent years, over 80 genetic loci related to T2D have been identified [?]. Nevertheless, the overall effect sizes of these loci account for less than 10% of the overall disease predisposition [?]. This poses the question of why, given that so much efforts has been directed at finding the genetic components of this disease, the loci found so far have such modest effects. This lack of large genetic effects do not only arise in T2D but also in almost all complex traits and could be explained by what is known as the "missing heritability" problem.

1.1.4 Missing heritability

We all know that "tall parents tend to have tall children", which is an informal way to say that height is a highly heritable trait. It is said that there are 30 cm from the tallest 5% to the shortest 5% of the population and genetics are accountable for 80% to 90% of this variation, which means that 27cm of variance are assumed to be "carried" by DNA variants from parents to offspring. Since 2010 the GIANT consortia has been investigating the genetic component of complex traits like height, body mass index (BMI) and waist to hip ratio (WHR). Even though they found many variants associated those traits, their findings only explain 10% of the phenotypic variance which corresponds to only a few centimeters in height [?].

In order to calculate heritability, we need to be able to measure it, so we need a formal definition. Heritability is defined as the proportion of phenotypic variance that is attributed to genetic variations. The total phenotypic variation is assumed to be caused by a combination of "environmental" and genetic variations $Var[P] = Var[G] + Var[E] + 2Cov[G, E]$ [?].

The environmental variance $Var[E]$ is the phenotypic variance attributable only to environment, that is the variance for individuals having the same genome $Var[E] = Var[P|G]$. Since cloning humans to calculate this term may be an overkill, we resort to approximate it based on phenotypic differences observed in monozygotic and dizygotic twins.

If the covariance factor $Cov[G, E]$ is assumed to be zero, we can define heritability as $H^2 = \frac{Var[G]}{Var[P]}$. This is called "broad sense heritability" because $Var[G]$ takes into account all possible forms of genetic variance:

$Var[G] = Var[G_A] + Var[G_D] + Var[G_I]$, where $Var[G_A]$ is the additive variance, $Var[G_D]$ is the variance form dominant alleles, and $Var[G_I]$ is the variance form interacting alleles (epistasis). Non-additive terms are difficult to estimate, so a simpler form of heritability called "narrow sense heritability" that only takes into account additive variance is defined as $h^2 = \frac{Var[G_A]}{Var[P]}$ [?].

Focusing on narrow sense heritability, the concept of "explained heritability" is defined as the part of heritability due to known variants with respect to all phenotypic variation ($\pi_{explained} = h^2_{known}/h^2_{all}$). Similarly, missing heritability is defined as $\pi_{missing} = 1 - \pi_{explained} = 1 - h^2_{known}/h^2_{all}$. When all variants associated with traits are known, then $\pi_{missing} = 0$.

Until recently, it was widely assumed by the research community that the problem of missing heritability lied in finding the appropriate genetic variants to account for the numerator of the equation (h^2_{known}) [?]. However, in a series of theorems published recently, it has been proposed that there is a problem in the way the denominator is estimated [?]. The authors created a limiting pathway model ($LP(k)$) that accounts for epistasis (gene-gene interactions) in k biological pathways. They showed that a severe inflation of h^2_{all} estimators occurs even for small values of k (e.g. $k \in [2, 10]$). As a result, genetic variants estimated to account only for 20% of heritability, could actually account for as much as 80% using an appropriate model [?].

Even though this result is encouraging, the problem is now shifted to detecting epistatic interactions, a problem that we analyze in section ?? and Chapter 4. In the same work [?], the authors show an example of power calculation assuming relatively large genetic effect that would require sequencing roughly $5,000$ individuals to detect links to genetic variants, which is a large but nowadays not uncommon, sample size. Nevertheless other estimates place the sample size requirements as high as $500,000$ individuals [?]. Even though this sounds as an extremely large number of samples, it is quickly becoming possible thanks to large technological advances and cost reductions in sequencing and genotyping technologies.

1.1.5 Conclusions

Although some genetic causes of complex traits, such as type II diabetes, have been found, only a small portion of the phenotypic variance can be explained. This might indicate that many risk variants are yet to be discovered. Recent studies on the topic of missing heritability report that these "difficult to find genetic variants" might be in epistatic interaction (analyzed in section **??**) or rare variants (see section **??**), analysis of either them requires more complex statistical models and larger sample sizes. In Chapter 4 of this thesis, we focus on methods for finding epistatic interactions related to complex disease and develop computationally tractable algorithms that can process data from sequencing experiments involving large number of samples in a reasonable amount of time.

1.2 Identification of genetic variants

Two of the main milestones in genetics were the discovery of the DNA structure in 1953 [?], followed by the first draft of the human genome in 2004 [?]. The cost of sequencing the first human reference genome was around $3 billion (unadjusted US dollars) and it was an endeavor that took around 10 years. Since that time, sequencing technology has evolved substantially so that a human genome can now be sequenced in a three days for a price of less than $1,000, according to prices estimated by Illumina, one of the main genome sequencer manufacturers.

Having a standard reference sequence facilitates comparisons and analysis. For most well known organisms, "reference genome" sequences are available and current large scale sequencing projects are extending significantly the number of genomes known, e.g. one project seeks to sequence 10,000 mammalian genomes [?], another is targeting all microbes that live within humans guts [?].

The amount of information delivered by sequencing devices is growing much faster than computer speed (Moore's law) and data storage capacity. Having to process huge amounts of sequencing information poses several challenges, a problem informally known as "data deluge". In the following sections, we explain how sequencing data is generated and how the huge amount of information delivered by a sequencer can be handled in order to make the problem tractable. Just as a crude example, a leading edge sequencing system is advertized to be capable of delivering 18,000 human genomes at $30x$ coverage per year, yielding over 3.2 PB of information. We want to transform this raw data into knowledge of genomic variants that contribute to disease risk with the ultimate goal to translate these risk variants into biological knowledge that can help to design drugs to treat or prevent disease. As expected, processing huge datasets consisting of thousands of sample is a complex problem. In Chapter 2 we show how mitigate or solve some of these issues, by designing a computer language specially tailored to tackle what are know as "Big data" problems.

1.2.1 Sequencing data

Different technologies for sequencing machines (or sequencers) exists. In a nutshell, a sequencer detects polymers (or chains) of DNA nucleotides and outputs a string of A, C, G, and Ts. Unfortunately, current technological limitations make it impossible to "read" a full chromosome as one long DNA sequence. Instead, modern sequencers produce a large number of "short reads", which range 100 bases to 20 Kilo-bases (Kb) in length, depending on the technology. Since sequencers are unable to read long DNA chains, preparing the

DNA for sequencing involves fragmenting it into small pieces. These DNA fragments are a random sub-samples of the original chromosomes. Reading each part of the genome several times allows to increase accuracy and ensure that the sequencer reads as much as possible of the original chromosomes. The coverage of a sequencing experiment is defined as the number of times each base of the genome is read on average. For instance, if the sequencing experiment is designed to produce one billion reads, and each read is 150 bases long, then the total number of bases read is 150Gb. Since the human genome is 3Gb, the coverage is said to be 50.

After sequencing a sample, we have millions of reads but we do not know where these reads originate from in the genome. This is resolved by aligning (also called mapping) reads to the reference genome, which is assumed to be very similar to the genome being sequenced. Once the reads are mapped, we can infer if the samples DNA has any differences with respect to the reference genome, a problem is known as "variant calling".

Using current technologies and computational methods for variant calling, detection accuracy varies significantly for different variant types. SNV are by far the most accurately detected. Insertions and deletions, collectively referred as InDels, can be detected less efficiently depending on their sizes. Small InDels consisting of ten bases or less are easier to detect than large InDels consisting of 200 bases or more. The reason being that the most commonly used sequencers reads DNA in stretches roughly 200 bases long. Due to this technological limitations, detection is less reliable for more complex variant types.

Although sequencing costs are dropping fast, it is still relatively expensive to sequence thousands of samples and in some cases it makes sense to focus on specific areas of the genome. A popular experimental setup is to focus on coding regions (exons). A technique called "exome sequencing" consists of capturing exons using a DNA chip and then sequencing the captured DNA fragments only. Exons are roughly 3% of the genome, thus this technique reduces sequencing costs significantly, for which it has been widely used by many research groups.

1.2.2 Sequence alignment

Given two sequences s_1 and s_2 from an alphabet (e.g. $\Sigma = \{A, C, G, T\}$), the alignment problem is to add gap characters ('-') to both sequences, so that a distance, such as Levenshtein distance, $d(s_1, s_2)$ is minimized.

This problem has a well known solution, the Smith-Waterman algorithm [**?**], which is a variation of the global sequence alignment solution from Needleman-Wunsch [**?**]. The main problem is that the algorithm is $O(l_1.l_2)$ where l_1 and l_2 are the length of the sequences. So, Smith-Waterman algorithm is slow for very long sequences, such as the human genome.

In order to speed up sequence alignments, several heuristic approaches emerged. Most notably, BLAST [**?**], which is used for mapping sequences several thousand nucleotides long (i.e. longer than a typical sequencer read) to a reference genome. BLAST uses an index to map parts of the query sequence, called seeds, to the reference genome. Once these seeds have been positioned against the reference, BLAST joins the seeds performing an alignment. Since the alignment is performed only using a small part of the reference, the algorithm is much faster.

1.2.3 Read mapping

Sequence alignment has an exact algorithm solution and several faster heuristic solutions. But even the fastest solutions are too slow to be used with the millions of reads generated in a typical sequencing experiment. Faster algorithms can be used if we relax our requirements in two ways: i) we allow for sub-optimal results, and ii) instead of requiring information of where each base of the read maps to the reference genome, we just want to know where the first base maps. This relaxed version of the alignment algorithm is called “read mapping” and the reduced complexity is enough to speed up the computations significantly. An implicit assumption in this formulation, is that the read will be very similar to the reference and that there will be no big gaps. Once the mapping is performed, the read is locally aligned, a strategy similar to BLAST algorithm [?].

Reformulating the problem this way, allows us to use other methods, such as suffix array [?]. Suffix arrays algorithms are fast, but memory requirements are $O[n\ log(n)]$ and this becomes the limiting factor. In order to reduce memory footprint of suffix arrays, Ferragina and Manzini [?] created a data structure based on the Burrows-Wheeler transform. This structure, known as an FM-Index, is memory efficient yet fast enough to allow mapping high number of reads. An FM-index for the human genome can be built in only 1Gb of memory, compared to 12Gb required for an equivalent suffix array [?]. Given a genome G and a read R, an FM-index search can find the N_{occ} occurrences of R in G in $O(|R| + N_{occ})$ time, where $|R|$ is the length of R [?].

Efficient indexing and heuristic algorithms can decrease mapping time considerably. Nevertheless, these algorithms are not guaranteed to find an optimal mapping. Several parameters, such as read length, sequencing error profile, and genome complexity profile can affect performance. The most commonly used implementation of the FM-index mapping algorithms are BWA [?, ?] and Bowtie [?, ?]. Each of them provide optimized versions for the two most common sequencing types: i) short reads with high accuracy [?, ?] or ii) longer reads with lower accuracy [?, ?].

It is worth noting that the mapping problem appears as a consequence of the technological limitations of sequencers. Having long, highly accurate reads, the problem becomes much easier to solve. As an extreme example, having only one read which is as long as a chromosome and has no errors requires no mapping processing.

1.2.4 Mapping quality

Sequencers not only provide sequence information, but also provide an error estimate for each base [**?**]. This is often referred as a quality (Q) value, which is the probability of an error, measured in negative decibels $Q = -10\ log_{10}(p)$.

Mapping quality is an estimation of the probability that a read is incorrectly mapped to the reference genome. Mapping algorithms provide estimates of mapping errors. In the MAQ model [**?**], which is one of the earliest models for calculating mapping quality, three main sources of error are explored: i) the probability that a read does not originate from the reference genome (e.g. sample contamination); ii) the probability that the true position is missed by the algorithm (e.g. mapping error); and iii) the probability that the mapping position is not the true one (e.g. if we have several possible mapping positions). It is assumed that the total error probability can be approximated as $\epsilon \approx max(\epsilon_1, \epsilon_2, \epsilon_3)$.

1.2.5 Variant calling

Once the sequencing reads have been mapped to the reference genome, we can try to find the differences between a sequenced sample and the reference genome. This is referred as “variant calling”. Several factors complicate this task, the two main ones being sequencing errors and mapping errors, described in **??**. Using sequencing and mapping error estimates, a maximum likelihood model can infer when there is a mismatch between a sample and the reference genome [?]. This method works best for differences of a single base (SNV), but it can also work with different degrees of success for short insertions or deletions (InDels) usually consisting of less than 10 bases.

Due to the nature of short reads, this family of methods does not work for structural genomic variants, such as large insertions, deletions, copy number variations, inversions, or translocations. A different family of algorithms are used to identify structural variants, but their accuracy so far has been low compared to SNV calling algorithm [?].

Aligning sequences that contain InDels (gaps) is more difficult than ungapped alignments since finding optimal gap boundary depends on the scoring method being used. This biases variant calling algorithms towards detecting false SNVs near InDels [?]. An approach to reduce this problem is to look for candidate InDels and perform a local realignment in those regions. This local re-alignment process reduces significantly the number of false positive SNVs [?]. Another approach to reduce the number of false positive SNVs calls near InDels involves the "Base Alignment Quality" (BAQ) [?], which is the probability of misalignment for each base. It can be shown that replacing the original base quality with the minimum between base quality and BAQ produces an improvement in SNV calling accuracy. The BAQ can be calculated using a special type of "Hidden Markov Model" (HMM) designed for sequence alignment [?, ?]. A more sophisticated option for reducing errors consist of performing a local genome re-assembly on each polymorphic region (e.g. HaplotypeCaller algorithm [?]).

Finally, the error probabilities inferred by the sequencers are far from perfect. Once the variants have been called, empirical error probabilities can be easily calculated [?] by comparing sequenced variants to a set of "gold standard variants" (i.e. variants that have been extensively validated). This allows to re-calibrate or re-estimate the error profile of the reads. This is know as a re-calibration step, and usually improves the number of false positives calls [?].

1.3 Functional annotations of genomic variants

Once DNA is sequenced, reads are mapped and variants are called as described in previous sections, variants identified are annotated in order to gain biological insight. For instance, we would like to know if it is located in a gene and if so whether the variant could be deleterious to the functionality of the protein encoded by the gene. This is the focus of Chapter 3 of my thesis.

The simplest case of a genetic annotation would be to know whether a genetic variant lies onto a gene or not. This would be trivial to calculate, since it only requires comparing the genomic location of the variant with the genomic location of known genes. However, in a sequencing experiment there are usually millions of variants and hundreds of thousands of genomic "features" such as genes, transcripts, exons, introns, splice regions, promoters, etc. The sheer volume of data requires time and memory efficient algorithms and data structures.

1.3.1 Functional annotations of coding variant

Genetic variants that are located within the coding region of a protein-coding gene are called coding variants. Although they form a small subset of all variants, they are the ones whose function can best be predicted. As explained by the central dogma of biology, genetic information flows from DNA molecules to mRNA molecules, which are used as a template to produce proteins.

A coding variant that produces a codon change is called "synonymous" or "non-synonymous" depending on whether the resulting amino acid remains the same of changes. If the variant is synonymous, we can be reasonably confident that there will be almost no effect in protein function, conversely if a non-synonymous variant creates a new STOP codon, it might be a strong indicator that protein function will be disrupted thus the variant is deleterious.

Estimating the putative effect of large coding variants (duplications, inversions or fusions) is much more challenging than in the cases of simple variants (SNVs, MNVs and small InDels) since there is still not enough studies to determine what effects large coding variants have in protein expression or function.

1.3.2 Non-coding annotations

For variants in non-coding regions of the genome, annotations are more difficult than in coding regions, mostly due to the fact that not only the location of most non-coding features (such as transcription factor binding sites, chromatin modifications, methylation) are not exactly known, but also non-coding features tend to be tissue specific. How DNA variants affect non-coding features is mostly speculative, and even for the few non-coding feature that have predictive models, these are riddled with false positives.

Assuming that non-coding features, or parts of them, have selective pressure to keep their functionality, conservation scores can be used as a proxy on how "important" these regions are. Nevertheless, this might not apply for certain classes of non-coding features, such as some transcription factors, where there is evidence of negative selection (meaning that transcription factors binding sites might not only not be conserved, but also change more rapidly than other genomic regions).

1.3.3 Conclusions

In Chapter 3 we show two software packages we designed for efficiently performing functional annotations of sequencing variants. These packages, SnpEff & SnpSift, allow to annotate, prioritize, filter and manipulate variant annotations as well as combine several public or custom-created databases. It should be noted SnpEff was one of the first annotation packages and has become one of the most widely used annotation software in both research and clinical environments.

1.4 Genome wide association studies

A genome wide association study aims at identifying genetic variants associated to a particular phenotype. First, the genomes (or exome, depending on the study design) of affected individuals (cases) and healthy individuals (controls) need to be sequenced, variants called, annotated and filtered. Then, the goal is to find variants that exhibit some statistical association with the trait or phenotype of interest, which could be a disease status (e.g. diabetes vs healthy), a biomedical measurement (e.g. cholesterol level), or any measurable characteristic (e.g. height). Since the genome is so large, patterns of mutations that suggest correlation may be encountered by chance, so we need to establish statistical significance in order to distinguish true association from spurious ones. Like most studies, we will focus on SNVs, but most methods can be extended to other genomic variants.

1.4.1 Single variant tests and models

Let's imagine that there is only one variant in the whole genome for the cohort we are analyzing. Since each individual has two sets of chromosomes, the variant can be present in one, both, or neither chromosomes. When a variant is in both chromosomes is said to be "homozygous", whereas if present in only one of the chromosomes, it is said to be "heterozygous". So the number of times a non-reference allele is present in an individual, is $N_{nr} = \{0, 1, 2\}$.

When the trait of interest is binary (e.g healthy vs disease), a cohort can be divided into cases and controls and we can build a 3 by 2 contingency table:

	$HomozygousReference$ $(N_{variant} = 0)$	$Heterozygous$ $(N_{nr} = 1)$	$Homozygousnon - reference$ $(N_{nr} = 2)$
$Cases$	$N_{ca,ref}$	$N_{ca,het}$	$N_{ca,hom}$
$Controls$	$N_{co,ref}$	$N_{co,het}$	$N_{co,hom}$

Further assumptions about how many variants are required to increase disease risk can reduce this 3×2 table to a 2×2 table. In the "dominant model", the effect of a mutated gene dominates over the healthy one, so one variant is enough to increase risk. The opposite, called "recessive model", is when both chromosomes have to be mutated in order to increase risk [**?**, **?**]. In these models, we can count how many cases and controls have at least one variant (dominant model) or two variants (recessive model). This simplifies the previous table, yielding a 2×2 contingency table, than can be tested using either a χ^2 test or a Fisher exact test [**?**].

Two other commonly used models, are the "multiplicative" and the "additive" models [**?**, **?**]. In these models, a disease risk is assumed to be multiplied (or increased) by a factor γ with every variant present. We cannot simplify the contingency table, so we assess significance using a Cochran-Armitrage test [**?**].

1.4.2 Multiple variant tests

In a real case scenario there are thousands or millions of variants. We can extend the concept shown in the previous section by performing individual tests for each variant present in the cohort. Multiple testing can be addressed either by performing a correction, such as False Discovery Rate [**?**, **?**], or using a stricter genome wide significance level. There are 3×10^9 bases in the genome, but taking into account the correlation between nearby variants (linkage disequilibrium), the genome wide significance level is generally accepted to be $p_{value} \leq 10^{-8}$.

In order to check if the null hypothesis of a significance tests is adequate, a QQ-plot is used (i.e. plotting the $y = -log(p_{value})$ vs $x = -log[rank(p_{value})/(N+1)]$, where N is the total number of variants). Adherence of the p-values to a 45 degree line on most of the range implies few systematic sources of association [**?**, **?**]. If the p-values have a higher slope than the $y = x$ line, there might be "inflation", possibly due to co-factors, such as population structure (see section **??**). If the inflation is not too high (e.g. less than 5%), this bias can be corrected by shifting the p-values towards the 45 degree slope. More sophisticated methods are explained in section **??**.

1.4.3 Continuous traits and correcting for co-factors

Methods analyzed so far are suitable for binary "traits" or "phenotypes" (e.g. disease vs. healthy individuals). Statistical methods that link genetic information to traits can also be used on continuous or "quantitative" traits (e.g. weight, height, cholesterol level, etc.). A linear regression can be used assuming the traits are approximately normally distributed [**?**, **?**]. A significance test (p_{value}) for linear models can be calculated using an F statistic, but more sophisticated methods are also available [**?**, **?**].

Using linear models, it is easy to include known co-factors to correct for biases or inflation. For instance, if it is known that a risk increases with age or that males are more susceptible than females, age and sex can be added to the linear equation in order to correct for these effects [**?**, **?**]. In a similar manner, we can add co-factors to binary traits using logistic regression.

1.4.4 Population structure

It is widely accepted that humans started in Africa and migrated to Europe, then to Asia and later to America [?]. Out of an initial population, a few individuals migrate and colonize a new territory. This implies that the genetic variety of the new colony is significantly reduced, compared to the previous population, since the genetic pool is only a small "founder population". The "Out of Africa" hypothesis implies that each new migration produced a reduction in genetic variety, also known as a "population bottleneck" [?].

As we previously mentioned, each individual inherits two chromosome sets, a maternal and a paternal one. In a process known as recombination, a chromosome that is formed by part of the maternal chromosome and part of the paternal one, is inherited to the offspring. As a result of recombination, a child has two sets of chromosomes that are one from each parent and, on average, half of a chromosome from each grandparent. This breaking and shuffling of chromosomes every generation, increases genetic diversity. Nevertheless if variants are located nearby in the chromosome, the chances that they are broken apart by recombination event are smaller than if they are further away from each other. This produces a correlation of close variants or "linkage disequilibrium" (LD). Nearby highly correlated variants are said to be in the same "LD-block" [?]. If a population has low genetic variety, the LD-blocks are large. So African population has more variety (smallest LD-blocks) and conversely, European, Asian and Amerindian populations have less variety (larger LD-blocks) [?].

1.4.5 Population as confounding variable

Imagine that we have a cohort of individuals drawn from two populations (P_A and P_B) and that individuals in P_A have much higher risk of diabetes than individuals from P_B. Now imagine that individuals from P_A have a variant v_A

more often, but v_A is actually neutral and has no health effects whatsoever. If we do not take into account population factors, our study would conclude that $variant_A$ is the cause of diabetes, just because we see $variant_A$ more often in affected individuals. In this case is clear that population structure is being a confounding variable. We could avoid this problem by analyzing each population separately [?], but this would cause a loss of statistical power since we have fewer samples.

A population that is a mixture of two or more populations, is known as an "admixed population". For instance the "African-American" population is a mixture of, roughly, 80% African and 20% European genomes [?, ?]. This means that analyzing a cohort of African-American individuals, we would get population structure as a confounding variable because of population admixture [?]. Obviously, in this case we cannot analyze each population separately, because each individual in the sample is a mixture of two populations.

The admixed population problem can be studied by performing a correction using the eigen-structure of the sample covariance matrix [?]. Samples can be arranged as a matrix C where each row is a sample and each column represents a position in the genome where there is a variant. The numbers $C_{i,j}$ in the matrix indicate whether a sample (row i) has a non-reference allele at a genomic position (column j). Since the allele can be present in zero, one, or two chromosomes in each individual, the possible values for $C_{i,j}$ are $\{0, 1, 2\}$. The covariance matrix is calculated as $M = \hat{C}^T.\hat{C}$, where $\hat{C}$ is the matrix C corrected to have zero mean columns. Usually, the first two to ten principal components of M are used as factors in linear models (see section ??) to correct for population structure [?].

Whether a cohort has any population structure and needs correction or not, can be tested using two methods: a) plotting the projections of the first two principal components and empirically observing the number of clusters in the chart, or b) using a statistic of the eigenvalues of M based on Tracy-Widom's distribution [**?**].

1.4.6 Common and Rare variants

The "allele frequency" (AF) is defined as the frequency a variant appears in a population. Variants are usually categorized according to AF into three groups: i) Common variants ($AF \geq 5\%$), "low frequency" ($1\% < AF < 5\%$), and iii) "rare variants" ($AF < 1\%$). Common variants originated earlier in the population while rare variants are either relatively recent or selected against.

There are three main models for disease susceptibility [**?**, **?**]:i) the Common-Disease-Common-Variant hypothesis (CDCV) assumes that if disease is common, it must be caused by a common variant; ii) the "infinitesimal hypothesis" proposes that there are many common variants each having small risk effects; and iii) the Common-Disease-Rare-Variant hypothesis proposes that there exists many rare variants, each one having large risk effects.

1.4.7 Rare variants test

The "rare variant model" assumes that multiple rare variants have large effects on a trait. The problem is that, since these variants are rare, huge sample sizes are required for tests to identify statistically significant associations. To overcome this problem, methods known as "burden tests", collapse several rare variants and perform statistical significance tests on grouped variants [**?**]. An example of collapsing technique is to count the number or rare variant in a given window and apply a Fisher exact test, as shown in section **??**. A limitation of some burden tests is that they implicitly assume that all rare variants have the same direction of effect, although rare variants might have no effect, be deleterious, or protective [**?**, **?**].

Several techniques allow weighting rare variants by collapsing them using a kernel matrix. This allows to incorporate other information, such as allele frequency and functional annotations. It can be shown that the statistic induced by kernel weighting functions follows a mixture of χ^2 distributions and there is an efficient way to approximate it [?, ?], avoiding computationally expensive permutations tests.

1.4.8 Conclusions

In this section we introduced the basic concepts and methodologies used in GWAS. Although fairly mature, there is still heavy research and continuous improvement on GWAS statistical methods. Not only it is well known that traditional (i.e. single marker) GWAS methods fail under non-additive models [?], but also variants so far discovered using these methods do not account for all the expected phenotypic variance attributed to genetic causes (i.e. missing heritability). As other authors pointed out, this might be because we need to look for epistatic variants which are not taken into account using these methods. In the next section, and in Chapter 4, we cover the topic of epistatic GWAS analysis.

1.5 Epistasis

Proteins are the most important part of the cell composing up to 50% of a cells dry weight compared to 3% of the DNA [?]. Proteins perform their functions mainly by interacting with other proteins, forming complex pathways that lead to a vast array of cellular functions including catalysis of chemical reactions, cell signaling, and structural conformation of the cell. The 3-dimensional structure of the protein, also called “tertiary structure”, is tailored to bind to other proteins in a specific manner to accomplish a functionality.

Genome wide association studies focus on single variants or nearby groups of variants. An often cited reason for the lack of discovery of high impact risk factors in complex disease is that these models ignore loci interactions [?] and recently they have been pointed out as a potential solution for the "missing heritability" problem [?]. With interactions being so ubiquitous in cell function, one may wonder why they have been so neglected by GWAS. We should point out that there are several reasons: i) models using interactions are much more complex [?] and by definition non-linear, ii) information on which proteins interacts with which other proteins is incomplete [?], iii) in the cases where there protein-protein interaction information is available, precise interacting sites are unknown [?]. Taking into account the last two items, we need to explore all possible loci combinations, thus the number of Nth order interactions grows as $O(M^N)$ where M is the number of variants [?]. This requires exponentially more computational power than single loci models. This also severely reduces statistical power, which translates into requiring larger cohort, thus increasing sample collection and sequencing costs [?].

In Chapter 4 we develop a computationally tractable model for analyzing putative interaction of pairs of variants from sequencing experiments involving large case / control cohorts of complex disease. Our model is based on combining multiple sequence alignments using a coevolutionary model in order to perform GWAS analysis of pairs of non-synonymous variants that may interact.

5.1 Detection of interacting sites in proteins using co-evolution

Proteins interactions and interaction loci are expensive to identify reliably experimentally and difficult to predict computationally. Some computational prediction methods are based on the assumption that protein interactions sites are under evolutionary pressure to avoid mutations [?], because such mutations reduce the efficiency or even disrupt pathways. Assuming that evolutionary pressure maintains favorable interaction between loci, compensatory mutations can happen more often than non-compensatory ones. The underlying idea is that fitness is higher for compensatory mutations and higher fitness co-occurring mutations would be fixed in the population. Since several organisms have been sequenced, we can try compare orthologous protein sites occurring in all these organisms and seek evidence of coevolution. It should be noted that this approach can be used to detect interacting sites within two different proteins or two interacting sites within the same protein. Detecting interacting sites within the protein can be valuable for determining protein structure. The most widely used method for inferring co-evolution starts from protein multiple sequence alignments ($\mathcal{M}_{sa}$), and identifies a pair of sites (e.g. one site from each interacting protein) that maximizes mutual information (MI) [?]. It is known that MI has some limitations [?] and is biased due to the fact that the multiple sequence alignment are related by an evolutionary process [?]. This means that some sequences will be very similar since they are evolutionarily close to each other (e.g. human and chimp), whereas other sequences will be very different (e.g. mouse and coelacanth).

A proper albeit more complex statistical analysis takes into account MSAs phylogenetic tree. Sophisticated coevolutionary models are usually designed with the intent of aiding protein structure predictions, which require to pinpoint the exact loci in each protein. These complex models can take anywhere from minutes to days to run for each pair of proteins, thus making them unfit for GWAS-scale analysis.

We propose to make use of co-evolutionary information to increase the interaction priors in a GWAS model. Since the goal is to increase GWAS priors instead of pinpointing the exact interaction loci, we can relax coevolutionary methods requirements to design computationally tractable models. In Chapter 4, we introduce an epistatic GWAS approach that while combining coevolutionary and sequencing information it is efficient enough to be applied to GWAS-scale, large cohort, datasets.

1.5.1 Epistatic GWAS

Arguably, the most common model linking binary phenotypes (disease vs. healthy) to genotypes is the logistic regression model which relates log odds probability of disease using multiple regression, $ln(\frac{p}{1-p}) = \hat{\beta}^T\hat{g}$, where p is the probability of disease, $\hat{g}$ are the models input variables (usually including genotype, sex, age, population structure, etc.), and $\hat{\beta}$ are the logistic regression coefficients.

Given a set of genotypes (typically genotype analysis includes 2,000,000 variants and tens of thousands of samples), the simplest way to look for interactions is an exhaustive search of all combinations. This raises two issues: i) multiple testing, which is often resolved by stringent significance threshold, and ii) computational feasibility, which is solved by efficient algorithms, parallelization, and heuristic approaches to quickly discard uninformative loci combinations.

The definition of epistasis, form a statistical perspective, is a "departure from a linear model" [?]. This means that in a logistic regression model the input includes terms with each of the genotypes (g_i and g_j), as well as an "interaction term" $gi.g_j$ [?]. Although we mainly talk about interaction between two loci, higher order interactions (three or more loci combinations) can be analyzed, but these models require more parameters and extremely large samples are required to accurately fit them.

Although a comprehensive review is out of the scope of this thesis, it is worth mentioning that several other approaches for epistatic GWAS exist. Here we mention a few (shown in alphabetic order):

- Allele frequency: In [?], an analysis of imbalanced allele pair frequencies is performed under the assumptions that an implicit test for fitness can be achieved looking for over/under-represented allele pairs in a given population.
- Bayesian model: In [?], a "Bayesian partitioning model" is used by providing Dirichlet prior distributions for each partition and computing posterior probabilities using Markov chain Monte Carlo (MCMC) algorithms. The methodology first test individual makers and picks only the top 10% to further investigate for epistasis, because it is prohibitive to test all loci.
- Linkage disequilibrium: Studying LD patterns in a population under two-loci model it was shown [?] that interactions creates LD in disease population. The authors show how LD-based p-values can uncover interaction and sometimes (in their simulations) outperform logistic regression tests.
- Machine learning: From a machine learning point of view, finding interacting variants is simply an optimisation and attribute selection procedure [?]. Several approaches have emerged to tackle the "interaction problem" and used a variety of different techniques [?, ?] , such as neural networks, cellular automata, random forests, multifactor dimensionality reduction, support vector machines, etc.

Although all these models have advantages under some assumptions, none of them seems to be a "clear winner" over the rest [?], thus currently there are no de-facto standards in epistatic analysis. In light of this, there is need of different approaches to be explored. In Chapter 5 we combine coevolutionary models and GWAS epistasis of pairs of putatively interacting loci, by using Bayes Factors to combine information.

1.6 Thesis roadmap and Contributions

The original research presented in this thesis covers topics related to the computational and statistical methodologies related to the analysis of sequencing variants to unveil genetic links to complex disease. Broadly speaking, we address three types of problems: (i) Data processing of large datasets from high throughput biological experiments such as resequencing in the context of a GWAS (Chapter 2); (ii) functional annotations, i.e. calculating variants impact at molecular, cellular or even clinical level (Chapter 3); (iii) identification of genetic risk factors for complex disease using models that combine population-level and evolutionary-level data to detect putative epistatic interactions (Chapter 4). It should be pointed out that the chapters are ordered similar to the analysis steps we used when analyzing our data for type II diabetes, starting from raw sequencing data and ending with GWAS analysis. When applicable, background material specific to each chapter is presented in a preface, together with an explanation of how that chapter ties in with the rest of the thesis.

This thesis comprises text and figures of scientific articles which have either been published, submitted for publication, or ready to be submitted (waiting upon data embargo restrictions):

Chapter 2 For this paper, PC conceptualized the idea and performed the language design and implementation. RS & MB helped in designing robustness testing procedures. PC, RS & MB wrote the manuscript.

- **Cingolani, Pablo**, Rob Sladek, and Mathieu Blanchette. "BigDataScript: a scripting language for data pipelines." Bioinformatics 31.1 (2015): 10-16.

Chapter 3 For this paper, PC designed, implemented and tested SnpEff & SnpSift. RS & MB suggested several extensions for common research use cases. PC, RS & MB wrote the manuscript. The manuscript was submitted to Nature Protocols, and the editor suggested for it to be published after the main T2D paper is accepted for publication (see next paragraph).

- **Cingolani, Pablo**, Rob Sladek, and Mathieu Blanchette. "Genomic variant annotation and prioritization" Ready for submission (waiting on consortia paper submission).

The following studies are T2D (type II diabetes) consortia projects which used SnpEff and SnpSift extensibly, several modules were designed with these projects in mind. This are part of a large consortia involving several institutions:

- McCarthy M., et al (T2D Genes Consortia). "Variation in protein-coding sequence and predisposition to type 2 diabetes", Ready for submission.
- Mahajan, Anubha, et al. "Identification and Functional Characterization of G6PC2 Coding Variants Influencing Glycemic Traits Define an Effector Transcript at the G6PC2-ABCB11 Locus." PLoS genetics 11.1 (2015): e1004876-e1004876.

The original SnpEff and SnpSift publications are provided in the appendices:

- **Cingolani, Pablo**, et al. "A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3." Fly 6.2 (2012): 80-92.

- **Cingolani, Pablo**, et al. "Using Drosophila melanogaster as a model for genotoxic chemical mutational studies with a new program, SnpSift." Toxicogenomics in non-mammalian species (2012): 92.

Chapter 4 For this paper, PC designed the methodology under the supervision of MB and RS. PC implemented the algorithms. PC, RS & MB wrote the manuscript. This work uses data from the T2D consortia, thus it cannot be published until the main T2D paper is accepted for publication (according to T2D data embargo).

- **Cingolani, Pablo**, Rob Sladek, and Mathieu Blanchette. "A co-evolutionary approach for detecting epistatic interactions in genome-wide association studies" Ready for submission (data embargo restrictions).

1.6.1 Other contributions

Other scientific articles (grouped by topic) published, submitted for publication, or ready to be submitted, not mentioned in this thesis:

Epigenetics

- **Cingolani, Pablo**, et al. "Intronic Non-CG DNA hydroxymethylation and alternative mRNA splicing in honey bees." BMC genomics 14.1 (2013): 666.
- Senut, Marie-Claude, et al. "Lead exposure disrupts global DNA methylation in human embryonic stem cells and alters their neuronal differentiation." Toxicological Sciences (2014).

- Ruden D., "Epigenetics as an answer to Darwins special difficulty Part 2: Natural selection of metastable epialleles in honeybee castes", Submitted.
- Arko S, et al. "Lead exposure induces changes in 5-hydroxymethylcytosine clusters in CpG islands in human embryonic stem cells and umbilical cord blood", Submitted.
- Senut, Marie-Claude, et al. "Epigenetics of early-life lead exposure and effects on brain development." Epigenomics 4.6 (2012): 665-674.

GWAS & Disease

- Oualkacha, Karim, et al. "Adjusted sequence kernel association test for rare variants controlling for cryptic and family relatedness." Genetic epidemiology 37.4 (2013): 366-376.
- Bongfen, Silayuv E., et al. "An N-ethyl-N-nitrosourea (ENU)-induced dominant negative mutation in the JAK3 kinase protects against cerebral malaria." PloS one 7.2 (2012): e31012.
- Hawn, Thomas R., et al. "Host-directed therapeutics for tuberculosis: can we harness the host?." Microbiology and Molecular Biology Reviews 77.4 (2013): 608-627.
- Meunier, Charles, et al. "Positional mapping and candidate gene analysis of the mouse Ccs3 locus that regulates differential susceptibility to carcinogen-induced colorectal cancer." PloS one 8.3 (2013): e58733.
- Caignard, Grgory, et al. "Genome-wide mouse mutagenesis reveals CD45-mediated T cell function as critical in protective immunity to HSV-1." PLoS pathogens 9.9 (2013): e1003637.

- Bouttier M., et al. “Genomics analysis reveals elevated LXR signaling reduces M. tuberculosis viability”, Submitted.
- Bouttier M., et al. “Genomic analysis of enhancers engaged in M. tuberculosis-infected macrophages reveals that LXR signaling reduces mycobacterial burden”, Submitted.

Other

- **Cingolani, Pablo**, and Jesus Alcala-Fdez. “jFuzzyLogic: a robust and flexible Fuzzy-Logic inference system language implementation.” FUZZ-IEEE. 2012.
- **Cingolani, Pablo**, and Jess Alcal-Fdez. “jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming.”International Journal of Computational Intelligence Systems

CHAPTER 2
BigDataScript: A scripting language for data pipelines

2.1 Preface

The overall goal in this thesis is to find genetic loci related to complex disease. In order to have enough statistical power to find these risk loci, we need to sequence thousands of cases and controls (i.e. patients and healthy individuals). Obviously the first step is to find all these patients, obtain patients consent, take samples and keep track of clinically relevant variables (such as age, sex, BMI, and glycemic traits). Just by the sheer number of patients involved, its easy to see that the logistics are challenging, to say the least.

Once the sequencing of each patients DNA is performed, we need to process the raw sequencing information by performing what is known as primary sequencing analysis, which involves mapping reads to the reference genome, calling variants, as well as performing several types of quality controls. The term primary analysis makes it sound as if this step is simple,but it is not. Managing such volume of information is a huge task that requires large computational resources, and coordinating the process involved at every stage of the analysis is not trivial, even if the jobs are relatively easy to parallelize.

As an example of the complexity and data volumes involved in these analysis pipelines, mapping the raw reads to the reference genome (i.e. the first stage of the primary analysis) for our T2D sequencing data is estimated to take over 12,000 CPU hours, that is over 32 CPU/years, under the most optimistic assumptions. At this magnitude hardware and failures become a significant issue since the probability of one or more nodes malfunction, while the data is being processed, is quite high.

We designed and implemented a simple script-like programming language called BigDataScript (BDS), with a clean and minimalist syntax to develop and manage pipeline execution and provide robustness to various types of software and hardware failures as well as portability. This programming language specifically tailored for data processing pipelines, improves abstraction from hardware resources and assists with robustness. Hardware abstraction allows BDS pipelines to run without modification on a wide range of computer architectures, from a small laptop to multi-core servers, server farms, clusters, clouds or even whole datacenters. BDS achieves robustness by incorporating the concepts of absolute serialization and lazy processing, thus allowing pipelines to recover from errors. By abstracting pipeline concepts at programming language level, BDS simplifies implementation, execution and management of complex bioinformatics pipelines, resulting in reduced development and debugging cycles as well as cleaner code. BDS was used to create data analysis pipelines required for our research, including the ones described throughout this thesis, and is currently used by other research groups and sequencing facilities in both academic and private environments.

The rest of the chapter is published in: Cingolani, Pablo, Rob Sladek, and Mathieu Blanchette. "BigDataScript: a scripting language for data pipelines." Bioinformatics 31.1 (2015): 10-16.

2.2 Introduction

Processing large amounts of data is becoming increasingly important and common in research environments as a consequence of technology improvements and reduced costs of high-throughput experiments. This is particularly the case for genomics research programs, where massive parallelization of microarray and sequencing-based assays can support complex genome-wide experiments involving tens or hundreds of thousands of patient samples [**?**]. With the democratization of high-throughput approaches and simplified access to processing resources (e.g. cloud computing), researchers must now routinely analyze large datasets. This paradigm shift with respect to the access and manipulation of information creates new challenges by requiring highly specialized skill, such as implementing data-processing pipelines, to be accessible to a much wider audience.

A data-processing pipeline, referred as “pipeline” for short, is a set of partially ordered computing tasks coordinated to process large amounts of data. Each of these tasks is designed to solve specific parts of a larger problem, and their coordinated outcomes are required to solve the problem as a whole. Many of the software tools used in pipelines that solve big data genomics problems are CPU, memory or I/O intensive and commonly run for several hours or even days. Creating and executing such pipelines require running and coordinating several of these tools to ensure proper data flow and error control from one analysis step to the next. For instance, a processing pipeline for a sequencing-based genome-wide association study may involve the following steps [**?**]: (i) mapping DNA sequence reads obtained from thousands of patients to a reference genome; (ii) identifying genetic changes present in each patient genome (known as “calling” variants); (iii) annotating these variants with respect to known gene transcripts or other genome landmarks; (iv) applying statistical analyses to identify genetic variants that are associated with differences in the patient phenotypes; and (v) quality control on each of the previous steps. Even though efficient tools exist to perform each of these steps, coordinating these processes in a scalable, robust and flexible pipeline is challenging because creating pipelines using general-purpose computer languages (e.g. Java, Python or Shell scripting) involves handling many low-level process synchronization and scheduling details. As a result, process coordination usually depends on specific features of the underlying systems architecture, making pipelines difficult to migrate. For example, a processing pipeline designed for a "multi-core server" cannot directly be used on a cluster because running tasks on a cluster requires queuing them using cluster-specific commands (e.g. qsub). Therefore, if using such a language, programmers and researchers must spend significant efforts to deal with architecture-specific details that are not germane to the problem of interest, and pipelines have to be reprogrammed or adapted to run on other computer architectures. This is aggravated by the fact that the requirements change often and the software tools are constantly evolving.

In the context of bioinformatics, there are several frameworks to help implement data-processing pipelines; although a full comparison is beyond the scope of this article, we mention a few that relate to our work: (i) Snakemake (Koster and Rahmann, 2012) written as a Python domain-specific language (DSL), which has a strong influence from `make` command. Just as in `make`, the workflow is specified by rules, and dependencies are implied between one rules input files and another rules output files. (ii) Ruffus (Goodstadt, 2010), a Python library, uses a syntactic mechanism based on decorations. This approach tends to spread the pipeline structure throughout the code, making maintenance cumbersome [?]. (iii) Leaf [?], which is also written as a Python library, expresses pipelines as graphs drawn using ASCII characters. Although visually rich, the authors acknowledge that this representation is harder to maintain than the traditional code. (iv) Bpipe [?] is implemented as a DSL on top of Groovy, a Java Virtual Machine (JVM)-based language. Bpipe facilitates reordering, removing or adding pipeline stages, and thus, it is easy for running many variations of a pipeline. (v) NextFlow (www.nextflow.io), another Groovy-based DSL, is based on data flow programming paradigm. This paradigm simplifies parallelism and lets the programmer focus on the coordination and synchronization of the processes by simply specifying their inputs and outputs.

Each of these systems creates either a framework or a DSL on a pre-existing general-purpose programming language. This has the obvious benefit of leveraging the languages power, expressiveness and speed, but it also means that the programmer may have to learn the new general-purpose programming language, which can be taxing and take time to master. Some of these pipeline tools use new syntactic structures or concepts (e.g. NextFlows data-flow programming model or Leafs pipeline drawings) that can be powerful, but require programming outside the traditional imperative model, and thus might create a steep learning curve.

In this article, we introduce a new pipeline programming language called BigDataScript (BDS), which is a scripting language designed for working with big data pipelines in system architectures of different sizes and capabilities. In contrast to existing frameworks, which extend general-purpose languages through libraries or DSLs, our approach helps to solve the typical challenges in pipeline programming by creating a simple yet powerful and flexible programming language. BDS tackles common problems in pipeline programming by transparently managing infrastructure and resources without requiring explicit code from the programmer, although allowing the programmer to remain in tight control of resources. It can be used to create robust pipelines by introducing mechanisms of lazy processing and absolute serialization, a concept similar to continuations (Reynolds, 1993) that helps to recover from several types of failures, thus improving robustness. BDS runs on any Unix-like environment (we currently provide Linux and OS.X pre-compiled binaries) and can be ported to other operating systems where a Java runtime and a GO compiler are available.

Unlike other efforts, BDS consists of a dedicated grammar with its own parser and interpreter, rather than being implemented on top of an existing language. Our language is similar to commonly used syntax and avoids inventing new syntactic structures or concepts. This results in a quick-to-learn, clean and minimalistic language. Furthermore, creating our own interpreter gives better control of pipeline execution and allows us to create features unavailable in general-purpose language (most notably, absolute serialization). This comes at the expense of expressiveness and speed. BDS is not as powerful as Java or Python, and our simple interpreter cannot be compared with sophisticated just-in-time execution or JVM-optimized byte-code execution provided by other languages. Nonetheless, in our experience, most bioinformatics pipelines rely on simple programmatic constructs. Furthermore, in typical pipelines, the vast majority of the running time is spent executing external programs, making the executing time of the pipeline code itself a negligible factor. For these reasons, we argue that BDS offers a good trade-off between simplicity and expressiveness or speed.

2.3 Methods

In our experience, using general-purpose programming languages to develop pipelines is notably slow owing to many architecture-specific details the programmer has to deal with. Using an architecture agnostic language means that the pipeline can be developed and debugged on a regular desktop or laptop using a small sample dataset and deployed to a cluster to process large datasets without any code changes. This significantly reduces the time and effort required for development cycles. As BDS is intended to solve or simplify the main challenges in implementing, testing and programming data processing pipelines without introducing a steep learning curve, our main design goals are (i) simple programming language; (ii) abstraction from systems architecture; and (iii) robustness to hardware and software failure during computationally intensive data analysis tasks. In the next sections, we explore how these concepts are implemented in BDS.

2.3.1 Language overview

BDS is a scripting language whose syntax is similar to well-known imperative languages. BDS supports basic programming constructs (if/ else, for, while, etc.) and modularity constructs such as functions and `include` statements, which are complemented with architecture-independent mechanisms for basic pipeline runtime control (such as task, sys, wait and checkpoint). At runtime, the BDS backend engine translates these high-level commands into the appropriate architecture-dependent instructions. At the moment, BDS does not support object-oriented programming, which is indeed supported by other pipeline tools based on libraries/DSL extending general-purpose programming languages. The complete language specification and documentation is available online at http://pcingola.github.io/BigDataScript.

Unlike most scripting languages, BDS is strongly typed, allowing detection of common type conversion errors at the initial parsing stage (pseudo-compilation) rather than at runtime (which can happen after several hours of execution). As the syntax of strict typing languages tends to be more verbose owing to longer variable declaration statements, we provide a type inference mechanism (operator `:=`) that improves code readability. For example (Listing 1), the variables `in` and `out` are automatically assigned the types the first time they are used (in this case, the type is assigned to be string).

2.3.2 Abstraction from resources

One of the key features of BDS is that it provides abstraction from most architecture-specific details. In the same way that high-level programming languages such as C or Java allow abstraction of the CPU type and other hardware features, BDS supports system-level abstraction, including the number and the type of computing-nodes or CPU-cores that are available to the pipeline and its component tasks, whether firing another process may saturate the servers memory or whether a process is executed immediately or queued.

Pipeline programming requires effective task management, particularly the ability to launch processes and wait for processes to finish execution before starting others. Task management can be performed using a single BDS statement, independently of whether this is running on a local computer or a cluster. Processes are executed using the task statement, which accepts an optional list of resources required by the task (for example, see Listing 1). The task consists of running a fictitious system command myProcess and diverting the output to `output.file`. BDS currently supports the following architectures: (i) local, single or multi-core computer; (ii) cluster, using GridEngine, Torque and Moab; (iii) server farm, using ssh access; and (iv) cloud, using EC2 and StarCluster. Depending on the type of architecture on which the script is run, the task will be executed by calling the appropriate queuing command (for a cluster) or by launching it directly (for a multi-core server).

Listing 2.1: `pipeline.bds` program. A simple pipeline example featuring and a maximum of 6 h of execution time (Line 5)

```
1 #!/usr/bin/env bds
2 in :=  input . file
3 out :=  output . file
4 task ( out <- in, cpus=2, timeout=6*HOUR ) {
5   sys myProcess $in > $out # Invoke command
6 }
```

BDS performs process monitoring or cluster queue monitoring to make sure all tasks end with a successful exit status and within required time limits. This is implemented using the `wait` command, which acts as a barrier to ensure that no statement is executed until all tasks finished successfully. Listing 2 shows a two-step pipeline with task dependencies using a `wait` statement (Line 13). If one or more of the `task` executions fail, BDS will wait until all remaining tasks finish and stop script execution at the `wait` statement. An implicit `wait` statement is added at the end of the main execution thread, which means that a BDS script does not finish execution until all tasks have finished running. It is common for pipelines to need multiple levels of parallel execution; this can be achieved using the `parallel` statement (or `par` for short). Wait statements accept a list of task IDs/parallel IDs in the current execution thread.

In addition to supporting explicitly defined task dependencies, BDS also automatically models implicit dependencies using a directed acyclic graph (DAG) that is inferred from information provided in the dependency operators (`<`) contained in `task` statements (see Listing 2, line 8). Finally, the `dep` expression defines a task whose conditions are not evaluated immediately (as it happens in `task` expressions) but only executed if required to satisfy a `goal`. Using `dep` and `goal` makes it easier to define pipelines in a `declarative` manner that is similar to other pipeline tools, as tasks are executed only if the output needs to be updated with respect to the inputs, independent of the intermediate results file, which might have been deleted.

2.3.3 Robustness

BDS provides two different mechanisms that help create robust pipelines: lazy processing and absolute serialization. When a processing pipeline fails, BDS automatically cleans up all stale output files to ensure that rerunning the pipeline will produce a correct output. If a BDS program is interrupted, typically by pressing Ctrl-C on the console, all scheduled tasks and running jobs are terminated or deallocated from the cluster. In addition to immediately releasing computing resources, a clean stop means that users do not have to manually dequeue tasks, which allows them to focus on the problem at hand without having to worry about restoring a clean state.

Lazy processing.. Complex processing pipelines are bound to fail owing to unexpected reasons that range from data format problems to hardware failures. Rerunning a pipeline from scratch means wasting days on recalculating results that have already been processed. One common approach, when using general-purpose scripting languages, is to edit the script and comment out some steps to save processing time, which is inelegant and error prone. A better approach is to develop pipelines that incorporate the concept of lazy processing [**?**], a concept popularized by the `make` command (Feldman, 1979) used to compile programs, and which simply means the work is not done a task invoking a fictitious command `myProcess` defined to require 2 CPUs twice. This concept is at the core of many of the pipeline programming tools, such as SnakeMake, Ruffus, Leaf and Bpipe. By design, when lazy processing pipelines are rerun using the same dataset, they avoid unnecessary work. In the extreme case, if a lazy processing pipeline is run on an already successfully processed dataset, it should not perform any processing at all.

BDS facilitates the creation of lazy processing pipelines by means of the dependency operator (`<-`) and conditional task execution (see Listing 1, line 5 for an example). The task is defined as `task (out < in)`, meaning that it is executed only if `out` file needs to be updated with respect to `in` file: for example, if `output.file` file does not exist, has zero length, is an empty directory or has been modified before `input.file`.

Absolute serialization.. This refers to the ability to save and recover a snapshot of the current execution state, compiled program, variables, scopes and program counter, a concept similar to continuations (Reynolds, 1993). BDS can perform an absolute serialization of the current running state and environment, producing checkpoint files from which the program can be re-executed, either on the same computer or on any other computer, exactly from the point where execution terminated. Checkpoint files (or `checkpoints` for short) also allow all variables and the execution stack to be inspected for debugging purposes (`bds -i checkpoint.chp`). The most common use of checkpoints is when a task execution fails. On reaching a `wait` statement, if one or more tasks have failed, BDS creates a checkpoint, reports the reasons for task execution failure and terminates. Using the checkpoint, pipeline execution can be resumed from the point where it terminated (in this case, at the most recently executed `wait` statement) and can properly re-execute pending tasks (i.e. the tasks that previously failed execution).

Limitations.. BDS is designed to afford robustness to the most common types of pipeline execution failures. However, events such as full cluster failures, emergency shutdowns, head node hardware failures or network problems isolating a subset of nodes may result in BDS being unable to exit cleanly, leading to an inconsistent pipeline state. These problems can be mitigated by a special purpose `checkpoint` statement that, as the name suggests, allows the programmer to explicitly create checkpoints. Given that the overhead of creating checkpoints is minimal (a few milliseconds compared with hours of processing time for a typical pipeline), carefully crafted checkpoint statements within the pipeline code can be useful to prevent losing processed data, mitigate damage and minimize the overhead when rerun, which can be critical for long running pipelines.

2.3.4 Other features

Here we mention some selected features that are useful in pipeline programming. Extensive documentation is available at http://pcingola.github.io/BigDataScript.

Automatic logging.. Logging all actions performed in pipelines is important for three reasons: (i) it helps debugging; (ii) it improves repeatability; and (iii) it performs audits in cases where detailed documentation and logging are required by regulatory authorities (such as clinical trials).

Listing 2.2: `pipeline_2.bds` program. A two-step pipeline with task dependencies. The first step (line 9) requires to run `myProcess` command on a hundred input files, which can be executed in parallel. The second step (line 19) processes the output of those hundred files and creates a single output file (using fictitious `myProcessAll` command). It should be noted that we never explicitly state which hardware we are using: (i) if the pipeline is run on a dual-core computer, as each process requires 2 CPUs, one `myProcess` instance will be executed at the time until the 100 tasks are completed; (ii) if it is run on a 64-core server, then 32 `myProcess` instances will be executed in parallel; (iii) if it is run on a cluster, then 100 `myProcess` instances will be scheduled and the cluster resource management system will decide how to execute them; and (iv) if it is run on a single-core computer, execution will fail owing to lack of resources. Thus, the pipeline runs independent of the underlying architecture. The task defined in line 18 depends on all the outputs from tasks in line 8 (`mainOut < outs`).

```
1 #!/usr/bin/env bds
2 // Step 1: Parallel processing of input files
3 string[] outs // Define a list of strings
4 for( int i=0 ; i < 100 ; i++ ) {
5     in  :=  input_$i . file
```

```
    out :=  output_$i . file
    task ( out <- in, cpus=2, timeout=6*HOUR ) {
        sys myProcess $in > $out
    }
    outs.add( out )         // Add all output files here
}
wait // Optional: Wait for all tasks to finish

// Step 2: Process all outputs from previous step
mainOut :=  main . txt
mainIn := outs.join(     ) // Create a string with all names
    (space-separated)
task ( mainOut <- outs, mem=10*G ) {
    sys myProcessAll $mainIn > $mainOut
}
```

Creating log files is simple, but it adds boilerplate code and increases the complexity of the pipeline. BDS performs automatic logging in three different ways. First, it directs all process StdOut/StdErr output to the console. Second, as having a single output can be confusing when dealing with thousands of processes running in parallel, BDS automatically logs each processs outputs (StdOut and StdErr) and exit codes in separate clearly identified files. Third, BDS creates a report showing both an overview and details of pipeline execution (Fig. **??**).

Automatic command line parsing.. Programming flexible data pipelines often involves parsing command-line inputsa relatively simple but tedious task.

BDS simplifies this task by automatically assigning values to variables specified through the command line. As an example, if the program in Listing 1 is called `pipeline.bds`, then invoking the program as `pipeline.bds -in another.file` will automatically replace the value of variable `in` with `another.file`.

Task re-execution.. Tasks can be re-executed automatically on failure. The number of retries can be configured globally (as a command-line argument) or by a task (using the `retry` variable). Only after failing `retry+1` times will a task will be considered to have failed.

2.3.5 BDS implementation

BDS is programmed using Java and GO programming languages. Java is used for high-level actions, such as performing lexical analysis, parsing, creating abstract syntax trees (AST), controlling AST execution, serializing processes, queuing tasks, etc. Low-level details, such as process execution control, are programmed in GO. As BDS is intended to be used by programmers, it does not rely on graphical interfaces and does not require installation of complex dependencies or Web servers.

Figure **??** shows the cascade of events triggered when a BDS program is invoked. First the script pipeline.bds (Fig. **??**A) is compiled to an AST structure (Fig. **??**B) using ANTLR (Parr, 2007). After creating the AST, a runnable-AST (RAST) is created. RAST nodes are objects representing statements, expressions and blocks from our BDS implementation. These nodes can execute BDS code, serializing their state, and recover from a serialized file, thus achieving absolute serialization. The script is run by first creating a scope and then properly traversing the RAST (Fig. **??**C). We note that if needed, this approach could be tuned to perform efficiently, as demonstrated by modern languages, such as Dart.

When recovering from a checkpoint, the scopes and RAST are deserialized (i.e. reconstructed from the file) and then traversed in recovery mode, meaning that the nodes do not execute BDS code. When the node that was executed at the time of serialization event is reached, BDS switches to run mode and the execution continues. This achieves execution recovery from the exact state at serialization time. Checkpoints are the full state of a programs instance and are intended as a recovery mechanism from a failed execution. This includes failures owing to corrupted or missing files, as BDS will re-execute all failed tasks when recovering, thus correcting outputs from those tasks. However, checkpoints are not intended to recover from programming errors, where the user modifies the program to fix a bug, as a previously generated checkpoint is no longer valid respect to the new source code.

When a task statement is invoked, process requirements, such as memory, CPUs and timeouts, can optionally be specified. Depending on the architecture, BDS either checks that the underlying system has appropriate resources (CPUs and memory) to run the process (e.g. local computer or ssh-farm) or relies on the cluster management system to appropriately allocate the task. If all task requirements are met, a script file is created (Fig. **??**D), and the task is executed by running an instance of bds-exec, a program that controls execution (Fig. **??**E). This indirection is necessary for five reasons, which are described in detail below: (i) process identification, (ii) timeout enforcement, (iii) logging, (iv) exit status report and (v) signal handling.

Process identification. means that bds-exec reports its process ID (PID), so that BDS can kill all child processes if the BDS script execution is terminated for some reason (e.g. the Ctrl-C key is pressed at the console).

Timeout enforcement. has to be performed by bds-exec as many underlying systems do not have this capability (e.g. a process running on a

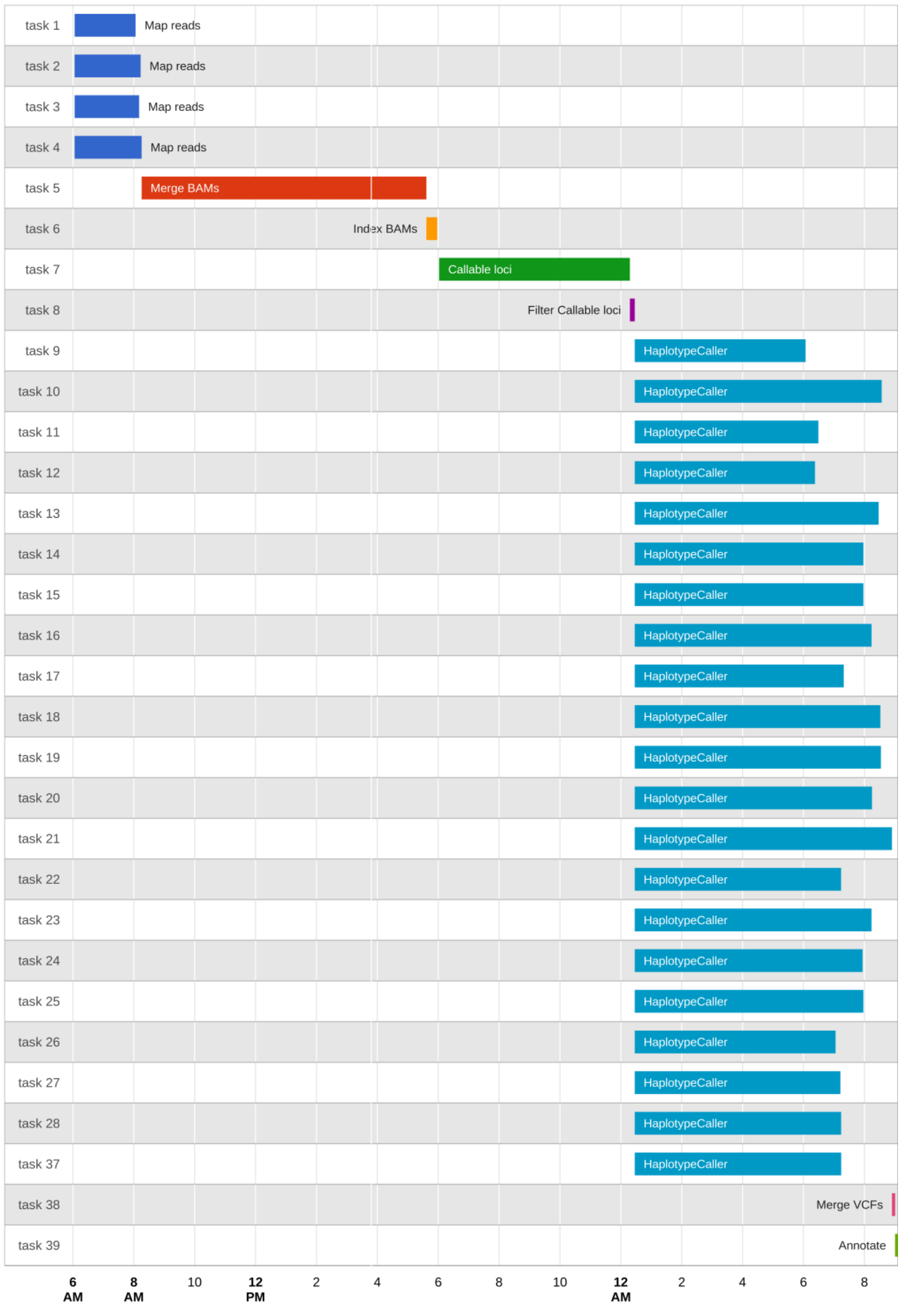


Figure 2–1: BDS report showing pipelines task execution timeline

server). When a timeout occurs, bds-exec sends a kill signal to all child processes and reports a timeout error exit status that propagates to the user terminal and log files.

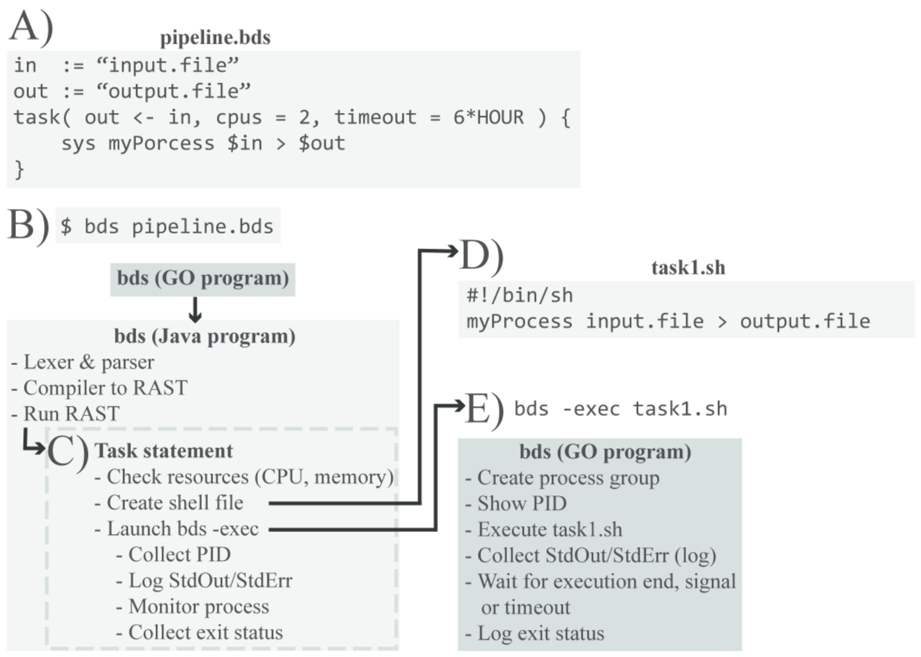
Logging. a process means that bds-exec redirects stdout and stderr to separate log files. These files are also monitored by the main BDS process, which shows the output on the console. As there might be thousands of processes running at the same time and operating systems have hard limits on the number of simultaneous file descriptors available for each user, opening all log files is not an option. To overcome this limit, BDS polls log file sizes, only opening and reading the ones that change.

Exit status. has to be collected to make sure a process finished successfully. Unfortunately, there is no unified way to do this, and some cluster systems do not provide this information directly. By saving the exit status to a file, bds-exec achieves two goals: (i) unified exit status collection and (ii) exit status logging.

Signal handling. is also enforced by bds-exec making sure that a kill signal correctly propagated to all subprocesses, but not to parent processes. This is necessary because there is no limit on the number of indirect processes that a task can run, and Unix/Posix systems do not provide a unified way to obtain all nested child processes. To be able to keep track of all subprocesses, bds-exec creates a process group and spawns the subprocess in it. When receiving a signal from the operating system, bds-exec traps the signal and propagates a kill signal to the process group.

2.4 Results

To illustrate the use of BDS in a real-life scenario, we present an implementation of a sequencing data analysis pipeline. This example illustrates



```
in  := “input.file”
out := “output.file”
task( out <- in, cpus = 2, timeout = 6*HOUR ) {
    sys myPorcess $in > $out
}
```

```
$ bds pipeline.bds
```

```
#!/bin/sh
myProcess input.file > output.file
```

```
bds -exec task1.sh
```

Figure 2–2: Execution example. (A) Script `pipeline.bds`. (B) The script is executed from a terminal. The GO executable invokes main BDS, written in JAVA, performs lexing, parsing, compilation to AST and runs AST. (C) When the task statement is run, appropriate checks are performed. (D) A shell script `task1.sh` is created, and a bds-exec process is fired. (E) bds-exec reports PID, executed the script `task1.sh` while capturing stdout and stderr as well as monitoring timeouts and OS signals. When a process finishes execution, the exit status is logged

three key BDS properties: architecture independence, robustness and scalability. The data we analyzed in this example consist of high-quality short-read sequences (200 coverage) of a human genome corresponding to a person of European ancestry from Utah (NA12877), downloaded from Illumina platinum genomes (http://www.illumina.com/platinumgenomes).

The example pipeline we created follows current best practices in sequencing data analysis [?], which involves the following steps: (i) map reads to a reference genome using BWA (Li and Durbin, 2009), (ii) call variants using GATKs HaplotypeCaller and (iii) annotate variants using SnpEff [?] and SnpSift [?]. The pipeline makes efficient use of computational resources by making sure tasks are parallelized whenever possible. Figure ?? shows a flowchart of our implementation, while the pipelines source code is available at `include/bio/seq` directory of our projects source code (https://github.com/pcingola/BigDataScript).

Architecture independence. . We ran the exact same BDS pipeline on (i) a laptop computer; (ii) a multi-core server (24 cores, 256 GB shared RAM); (iii) a server farm (5 servers, 2 cores each); (iv) a 1200-core cluster; and (v) the Amazon AWS Cloud computing infrastructure (Table ??). For the purpose of this example and to accommodate the fact that running the pipeline on a laptop using the entire dataset would be prohibitive, we limited our experiment to reads that map to chromosome 20. The architectures involved were based on different operating systems and spanned about three orders of magnitude in terms of the number of CPUs (from 4 to 1200) and RAM (from 8GB to 12TB). BDS can also create a cluster from a server farm by coordinating raw SSH connections to a set of computers. This minimalistic setup only requires that the computers have access to a shared disk, typically using NFS, which is a common practice in companies and university networks.

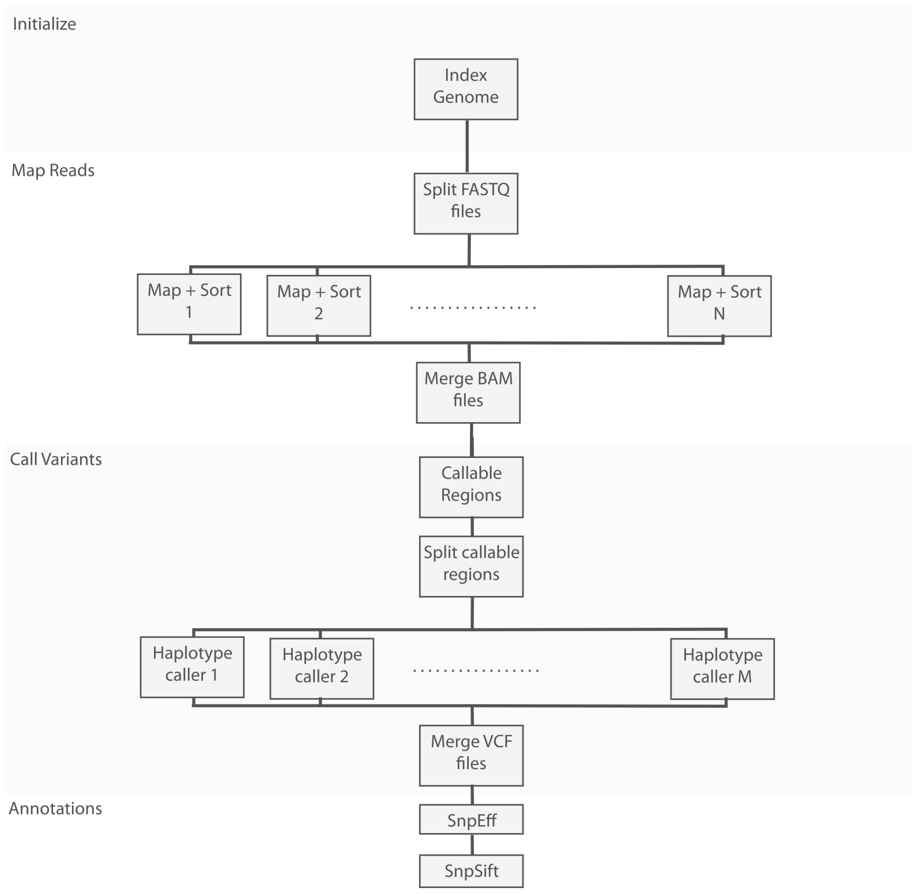
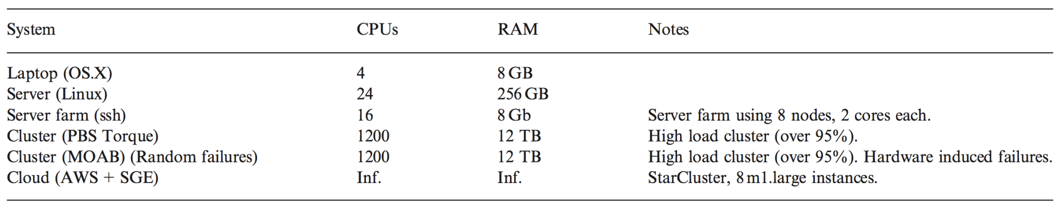


Figure 2–3: Whole-genome sequencing analysis pipelines flow chart, showing how computations are split across many nodes



System	CPUs	RAM	Notes
Laptop (OS.X)	4	8 GB	
Server (Linux)	24	256 GB	
Server farm (ssh)	16	8 Gb	Server farm using 8 nodes, 2 cores each.
Cluster (PBS Torque)	1200	12 TB	High load cluster (over 95%).
Cluster (MOAB) (Random failures)	1200	12 TB	High load cluster (over 95%). Hardware induced failures.
Cloud (AWS + SGE)	Inf.	Inf.	StarCluster, 8 m1.large instances.

Figure 2–4: Architecture independence example. Notes: Running the same BDS-based pipeline, a sequence variant calling and analysis pipeline, on the same dataset (chr20) but different architectures, operating systems and cluster management systems.

In all cases, the overhead required to run the BDS script itself accounted for 52 ms per task, which is negligible compared with typical pipeline runtimes of several hours.

Robustness.. To assess BDSs robustness, we ran the pipeline on a cluster where 10% of the nodes have induced hardware failures. As opposed to software failures, which are usually detected by cluster management systems, hardware node failures are typically more difficult to detect and recover from. In addition, we elevated the cluster load to 495% to make sure the pipeline was running on less than ideal conditions. As shown in Table **??**, the pipeline finished successfully without any human intervention and required only 30% more time than in the ideal case scenario because BDS had to rerun several failed tasks. This shows how BDS pipelines can be robust and recover from multiple failures by using lazy processing and absolute serialization mechanisms.

Scalability.. To assess BDSs scalability, we ran exactly the same pipeline on two datasets that vary in size by several orders of magnitude (Table **??**): (i) a relatively small dataset (chromosome 20 subset, 2GB) that would typically be used for development, testing and debugging and (ii) a high-depth whole-genome sequencing dataset (over 200 coverage, roughly 1.5 TB).

Dataset	Dataset size	System	CPUs	RAM
chr20	2 GB	Laptop (OS.X)	4	8 GB
Whole genome	1.5 TB	Cluster (MOAB)	22 000	80 TB

Figure 2–5: Scaling dataset sized by a factor of 1000. Notes: The same sample pipeline run on dataset of 2 GB (reads mapping to human chromosome 20) and 1.5 TB (whole-genome data set). Computational times vary according to systems resources, utilization factor and induced hardware failures.

2.5 Discussion

We introduced BDS, a programming language that simplifies implementing, testing and debugging complex data analysis pipelines. BDS is intended to be used by programmers in a similar way to shell scripts, by providing glue for several tools to ensure that they execute in a coordinated way. Shell scripting was popularized when most personal computers had a single CPU and clusters or clouds did not exist. One can thus see BDS as extending the hardware abstraction concept to data-center level while retaining the simplicity of shell scripting.

BDS tackles common problems in pipeline programming by abstracting task management details at the programming language level. Task management is handled by two statements (`task` and `wait`) that hide system architecture details, leading to cleaner and more compact code than general-purpose languages. BDS also provides two complementary robustness mechanisms: lazy processing and absolute serialization.

A key feature is that being architecture agnostic, BDS allows users to code, test and debug big data analysis pipelines on different systems than the ones intended for full-scale data processing. One can thus develop a pipeline on a laptop and then run exactly the same code on a large cluster. BDS also provides mechanisms that eliminate many boilerplate programming tasks, which in our experience significantly reduce pipeline development times. BDS can also reduce CPU usage, by allowing the generation of code with fewer errors and by allowing more efficient recovery from both software and hardware failures. These benefits generally far outweigh the minimal overhead incurred in typical pipelines.

CHAPTER 3
SnpEff: Genomic variant annotation and prioritization

3.1 Preface

As this thesis is focused on extracting biological insight from sequencing data, in this chapter we examine algorithms we created for calculating "functional annotations" of genomic variants. In essence, functional variant annotations are bits of biological knowledge that allow us to make prioritize variants that are assumed to be more relevant to the phenotypic trait under study and to filter out variants assumed irrelevant. The spectrum of functional annotations for a genomic variant is wide and may involve information on which genes are affected by the variant, how the protein product is affected, how conserved is the genomic region the variant lies onto, and which clinically relevant information is associated with the loci; just to mention a few typical use cases.

When trying to find variants that affect risk of complex disease, statistical power is paramount. We need to be able to "separate wheat from chaff". In our context this means two different but closely related tasks: i) performing functional annotations, and ii) using that information for prioritizing variants (and filtering out the ones we suspect are not related to the particular trait under study). Failing to efficiently filter out irrelevant variants would reduce our statistical power as more statistical tests are calculated, thus would decrease our chances of finding the associations we are looking for. In order to efficiently annotate and filter variants, we created two software packages called SnpEff and SnpSift that deal with the annotation and filtering aspects respectively.

3.2 Epilogue?

At the beginning of my Ph.D., functional annotation of genomic variants was an unsolved problem with many research labs creating in-house custom solutions that oftentimes were inefficient and lacking of rigorous testing. As a consequence, shortly after SnpEff & SnpSift were released they quickly became widely adopted by the research community as well as many private organizations. Currently SnpEff & SnpSift has over 250 downloads per week (as reported by SourceForge, where the tools are hosted). So far SnpEff & SnpSift have been cited over 400 times.

3.2.1 Data structures for annotations

A very simple approach used by ANNOVAR [**?**] is to create an index by dividing each chromosome into N bins of equal size. All genomic features are stored in a hash table indexed by chromosome name and bin number. This approach has running time of $O(n)$ where n is the number of features, but it can be easily tuned by creating small bins, at the cost of increased memory requirements.

Another approach [**?**] is to use an "interval forest", which is a hash of "interval trees" indexed by chromosome. Each interval tree is composed of nodes. Each node has five elements i) a center point, ii) a pointer to a node having all intervals to the left of the center, iii) a pointer to a node having all intervals to the right of the center, iv) all intervals overlapping the center point sorted by start position, and v) all interval overlapping the center point sorted by end position. Querying an interval tree requires $O[log(n) + m]$ time, where n is the number of features in the tree and m is the number of features in the result. Having a hash of trees optimizes the search by reducing the number of intervals per tree.

3.3 Background

The development of cost-effective, high-throughput next generation sequencing (NGS) technologies is poised to have a profound impact on our ability to study the effects of individual genetic variants on the pathogenesis and progression of both monogenic and common polygenic diseases. As sequencing costs decrease and throughput increases, it has now become possible to quickly identify a large number of sequence polymorphisms (SNVs, indels, structural) using samples from affected and unaffected subjects and investigate these in epidemiologic studies to identify genomic regions where mutations increase disease risk. However, translating this information into biological or clinical insights is challenging as it is often difficult to determine which specific polymorphisms are the main pathogenetic drivers of disease across a population; and more importantly, how they affect the activity of disease-related molecular pathways in tissues and organism a specific patient. In part, this difficulty results from the large number of genetic variants that are observed in individual genomes (the human population is believed to contain approximately 3.5 million polymorphic sites with minor allele frequency above 5%) combined with the limited ability of computational approaches to distinguish variants with no impact on genome function (the vast majority) from variants affecting gene function or expression that may be associated with disease risk or drug response (the minority). The development of algorithms for automated variant annotation,which link each variant with information that may help predict its molecular and phenotypic impact, is a critical step towards prioritizing variants that may have a functional impact from those that are harmless or have irrelevant functional effects. The goal of this protocol is to collect relevant information that will help answer questions about genetic variants discovered in next-generation sequencing studies, including: (i) will a given coding variant affect the ability of a protein to carry its functions; (ii) will a given non-coding variant affect the expression or processing of a given gene; and ultimately (iii) will a given coding or noncoding variant have any impact on phenotypes of interest?

Answering these questions is essential for many types of analyses that use large-scale genomics datasets to study quantitative traits and diseases, particularly when only a small number of individuals is studied comprehensively at a genome-wide level. For example, most genome-wide association studies (GWAS) or exome sequencing studies lack the statistical power to identify rare variants or variants with small effects associated with a disease, in part due to the large number of variants assayed. This limitation can be addressed by directing subsequent experimental steps to focus on smaller sets of genetic variants that have been prioritized based on external evidence of their putative impact. The common impairment of DNA repair mechanisms and chromatin stability in malignant cells leads to a similar challenge in cancer genomics, where the hundreds or thousands of mutations that distinguish an individuals tumor and germline genomes need to be classified on the basis of their putative phenotypic effects and potential roles in carcinogenesis.

The large number of databases containing potentially helpful information about a given variant make the process of gathering and presenting relevant data challenging, despite excellent tools that already exist to analyze large genomics datasets (including GATK2 and Galaxy3) and visualize the results (such as the UCSC4 or Ensembl5 genome browsers). Each of these databases uses its own format and is updated asynchronously, which makes it difficult for any analysis to remain up to date. In addition, the lack of comprehensive and computationally efficient models that allow integrative analyses using these resources, makes the task of comprehensive variant annotation overwhelming.

By efficiently combining information from tens or hundreds of genome-wide databases, the tools and protocol described here are designed to greatly facilitate the process of variant annotation, and make it accessible to groups with limited bioinformatics expertise or resources.

In this protocol, we describe an approach to variant annotation that automatically collects, integrates, and presents a wide body of publicly available evidence of functional impact of a given set of genomic variants. The pipeline, based on the SnpEff package6, is easy to execute and efficiently extracts a comprehensive set of variant annotations that can be used to prioritize downstream clinical or functional studies. SnpEff is used in many large genome centers and supports variant annotation for thousands of species, although the extent and quality of annotations extracted for a set of variants depends on the amount of publicly available genomics data for that species. In the case of whole-genome or exome sequences from human DNA samples, SnpEff extracts metadata (annotations) for each variant from relevant sources including gene annotation data identifying transcribed and translated regions; estimates of the frequency with which each variant occurs in different populations (from the 1000 Genomes project and the Exome Sequencing project1); and data that describes the function of regulatory elements that may be altered by the variant (obtained from the ENCODE project7 and Epigenome Roadmap8). SnpEff allows flexible and efficient querying of these annotations and is sufficiently fast to analyze very large sets of variants on a small computer (see Box 1 for computational and algorithmic considerations). It is also able to detect and be robust to a variety of gene annotation inconsistencies that would otherwise trigger false-positive high-impact variant predictions. In addition to SnpEff, an number of other annotation packages have been developed (including ANNOVAR9, the Ensembl Variant Effect Predictor10, GEMINI11 and

VAT12), which differ in terms of their functionality, ease-of-use, computational efficiency, and robustness.

3.4 Genetic variants

For most species, genetic variants are identified by comparing genome sequences from an individual organism to a reference (haploid) genome (see Box 2). A genetic variant, at a specific location in the genome (genetic locus) can be represented by agenotype which describes the difference between the DNA sequence present on each chromosome in the individuals diploid genome and the reference genome. At a specific polymorphic locus, an individual can thus be homozygous for the reference allele, heterozygous, or homozygous for the non-reference allele.

There are as many types of genetic variants as there are types of mutations, and their frequency and breadth of impact on the genome vary tremendously13. The most common type of variant identified by current technologies and analysis approaches is a single base difference with respect to the reference genome. Depending on whether the variant was identified in an individual or in a population, it is called a Single Nucleotide Variant (SNV) or Single Nucleotide Polymorphism (SNP). Sequence differences affecting several consecutive nucleotides are called a multiple nucleotide polymorphism (MNP) and are typically treated as a single variant locus if they are in perfect linkage disequilibrium. Short insertions and deletions (indels) of a chromosome region range from 1 to 20 bases in length are approximately 30 times less frequent than SNVs1but may have profound effects on protein activity by altering the translation reading frame or deleting a protein domain. Genomic variants involving larger regions are more rare and more difficult to infer using short-read NGS technologies. Those include large deletions, which can result in the loss of an exon or one or more whole genes, and insertions, which often originate from transposable elements or tandem duplications. Large genome variants that cause the number of copies of a particular genomic region to be polymorphic in a population are called a Copy Number Variants (CNV). Genomic rearrangements such as inversions and translocations are events that involve two or more genomic breakpoints and a reorganization of genomic segments, possibly resulting in gene fusions or loss of critical regulatory elements. This protocol does not address the annotation or rearrangements due to the challenges involved in their identification and functional characterization and their relative rarity in the germ line.

The process of inferring variants present in an individuals genome from sequencing data is called variant calling and is based on sophisticated algorithms that have been reviewed elsewhere14. Genome-wide variant calling has until recently largely been done using genotyping arrays (for SNVs) or Comparative Genomic Hybridization arrays (for CNVs). The inherent limitations of these technologies, particularly their ability to only assay genotypes at sites that are known in advance to be polymorphic, combined with the declining cost of sequencing, have now made approaches based on high-throughput resequencing the tool of choice for variant calling in clinical studies. Although this is a challenging task and remains an important area of research, many high-quality tools exist for calling SNVs and indels (such as GATK2 and SamTools15), as well as detecting CNVs (such as PennCNV16 and CNVhap17), and structural variants (e.g. VariationHunter18). The output of these tools variant calls are stored using the standardized format called the Variant Call Format19 (VCF; see Box 3). Their calling accuracy depends on the type of variants, their frequency in the population of patients or tumor cells being studied, and the quantity (coverage) and quality of the sequencing data. As the length, accuracy and coverage of sequencing reads increases, variant calling will become easier and more accurate. Therefore, we discuss here the problem of annotating the variants identified by some of these tools, and refer the reader to the review by Nielsenet al.14 to learn more about the process of variant calling itself.

3.4.1 Types of genetic annotations

The process of genetic variant annotation consists of the collection, integration, and presentation of experimental and computational evidence that may shed light on the impact of each variant on gene or protein activity and ultimately on disease risk or other phenotypes. Variant annotation has traditionally been divided in two apparently independent but actually interrelated tasks based on the variants location with respect to known protein-coding genes (see Table 1 for a list of commonly used variant annotations).Coding variant annotation focuses on variants that are located within coding regions of annotated protein-coding genes and attempts to assess their impact on the function of the encoded protein. In contrast,non-coding variant annotation focuses on variants located outside the coding portion of genes (i.e. in intergenic regions, UTRs, introns, or non-protein-coding genes) and aims to assess their potential impact on transcriptional and post-transcriptional gene regulation. These two categories of variant annotations are not mutually exclusive, as variants located within exons can often have an impact on the gene transcripts processing (splicing). In addition, some transcripts can have both protein-coding and non-coding functions. Despite the intermingling of the notion of coding and noncoding variants, we will consider each type of annotation separately as assessing their impact requires different sources of data and algorithms.

The ultimate goal of variant annotation is to predict the impact of a sequence variant, although this is an ill-defined term. One the one hand, one may be interested in the molecular impact of a variant on the activity of a protein. On the other, others may be interested in a variants impact on much higher-level phenotypes such as disease risk. Mutations that are predicted to completely abrogate a genes activity are calledloss-of-function (LOF) mutations; while mutations that are tentatively predicted to have less severe consequences are called moderate or low impact mutations.In practice, a variant will be predicted to cause LOF if it has two properties: (i) its molecular impact is reliably predictable by existing computational approaches (e.g. gain of stop-codon); and (ii) its functional impact, reflected by altered protein activity or expression levels, is expected to be large. Many types of variants, including most non-coding variants, may have a large functional impact but lack predictability, and as a consequence are typically not predicted to be LOF variants.

3.4.2 Coding variant annotation

Coding variants occur in a translated exon. When a reliable gene annotation is available, their main impact can be classified by determining their effect on the translated amino acid sequence (if any). A synonymous coding variant (also called silent) does not change the sequence of amino acids encoded by the gene, although it may impact aspects of post-transcriptional regulation such as splicing and translation efficiency and can affect the total protein activity through changes in the amount of translated protein that is made in the cell. In contrast, a non-synonymous coding variant changes one or more amino acids encoded by the gene and can directly alter the proteins activity, localization or stability. Non-synonymous variants include missense substitutions that change a single amino acid, nonsense substitutions that lead to the gain of a stop codon,frame-preserving indels that insert or delete one or more amino acids, and frame-shifting indels that may completely alter the proteins amino acid sequence. Primary annotation and assessment of impact, which performed directly by SnpEff, determines whether a variant falls in any of these categories.

Caveats

Gene misannotation. Genomic variants that have a significant effect on a proteins expression or function represent a very small fraction of all variants. Assembly and gene annotation errors or genomic oddities that break classical computational models are also rare, but often overestimate the variants impact. This implies that one is likely to find a non-negligible fraction of false-positive high-impact variants among the list of what appear to be the strongest candidates for variants with severe effects. Tools such as SnpEff can anticipate some of the most common causes of misannotation, but the number and diversity of the type of events that can lead to false-positives makes the task very challenging. As a consequence, one should always manually inspect the top candidates to ensure that they have been assigned to the correct genes and transcripts.

Gene isoforms. In higher eukaryotes, most genes have more than one transcript (or isoform), due to alternative promoters, splicing, or polyadenylation sites. For example, a human gene has an average of 8.8 annotated messenger RNA (mRNA) isoforms and some genes are believed to have over 4,000 isoforms resulting from complex splicing programs. For these genes, a variant may be coding with respect to one mRNA isoform and non-coding with respect to another. There are two frequent approaches to address this situation: (i) annotate a variant using the most severe functional effect predicted for at least one mRNA isoform; or (ii) use only a single canonical transcript per gene to perform primary annotation.

Variant calling for indels. Variant annotation relies on knowing the exact genomic coordinates of the variant: this is rarely a problem for isolated SNVs; however, insertions and deletions often cannot be located unambiguously. Consider for example the variant AA -¿ A. This mutation results in the loss of a single base, but was it the first or second A that was deleted? From the standpoint of the cell, this question is irrelevant and deletion of any A will have the same effect. In contrast, from the standpoint of most variant annotation software, deleting the first A is different from deleting the second. Consider the scenario of a previously annotated transcript where the first A is part of the 5 UTR and the second is the first base of a start codon. If the missing base is assigned to the leftmost position in the motif (as is the current convention), the deletion would be annotated as a low impact 5UTR variant. However, assigning it to the rightmost A would make it appear (incorrectly) to be a high-impact start-codon deletion. Similar issues may arise when considering conservation scores or transcription factor binding site (TFBS) predictions.

3.4.3 Loss of function variants

True LOF variants are difficult to predict computationally, but specific types of genetic changes will frequently lead to severely impaired protein activity. These include (i) stop-gains (nonsense mutations) and start-loss; (ii) indels causing frameshifts; (iii) large deletions that remove either the first exon or at least 50% of the protein coding sequence; and (iv) loss of splice acceptor or donor sites that alter the protein-coding sequence. Variants that introduce premature in-frame stop codons (nonsense mutations and most frameshift indels) are expected to abolish protein function, unless the variant is very near the C-terminus of the coding region20 (effectively, downstream of the last functional domain in the protein). This may cause severe consequences in affected cells, tissues or organism, as is seen for mutations that cause monogenic diseases21. In addition, a new stop codon that lies upstream of the last exon will likely trigger nonsense mediated decay (NMD), a process that degrades mRNA before protein synthesis occurs22. NMD predictions are not exact and many factors can affect mRNA degradation, including the variants distance from the last exon-exon junction or poly-A tail, and the possibility that transcription may re-initiate downstream of the LOF variant23.

A variant that leads to the loss of a stop codon, sometimes called aread-through mutation, will result in an elongated protein-coding transcript that terminates at the next in-frame stop codon. While there are no general models that predict how deleterious this may be, variants that elongate the reading frame can also result in aberrant folding and degradation of the nascent proteins, leading to activation of cellular stress response pathways in addition to their direct effects on protein activity and expression levels21.

The effect of the loss of a start codon depends on the location of a replacement start codon with respect to the translation start site and reading frame of the native protein. If the new start codon maintains the reading frame, the only consequence may be the loss of a few amino acids in the protein transcript; however, in many cases, the new start codon will not be in-frame, thus producing a frame-shifted protein that is later degraded. In addition, the new start codon may lack an appropriate regulatory context (for example, if there is no Kozak sequence nearby or if it disrupts 5 UTR folding) leading to reduced expression of an N-terminally truncated protein. Consequently, losing a start codon is thought to be highly deleterious in most cases, due to the potential that it may reduce both protein production and activity.

Caveats

Rare amino acids. Through a process called translational recoding, a UGA "Stop" codon located in the appropriate mRNA context (determined by both primary mRNA sequence and secondary structure) may be translated to incorporate a selenocysteine amino acid (Sec / U). In humans, this occurs at approximately 100 codons located in mRNAs whose 3 UTR contains a Selenocysteine insertion sequence element (SECIS). Since the translation machinery goes so far to encode these special rare amino acids, the expectation is that mutations at those sites would be highly deleterious. This is supported by evidence that reduced efficiency of selenocysteine incorporation is linked to severe clinical outcomes, such as early onset myopathy 24 and progressive cerebral atrophy 25.

False-positives in LOF predictions. Variants predicted to result in a LOF sometimes actually produce proteins that are partially functional 26. In fact, an apparently healthy individual is typically heterozygous for around 100 predicted LOF variants, and homozygous for roughly 10 variants, but many of those are unlikely to completely abolish the protein function. Indeed, these variants are enriched toward the 3 end of the gene, where they are likely to be less deleterious.

3.4.4 Variants with low or moderate impact

Compared to the high impact variants discussed above, where extensive prior biological evidence strongly suggests that a specific type of variant will severely impair protein activity, there are few guidelines that can reliably predict how the majority of nonsynonymous (missense) variants will alter protein function or expression. As a result, the primary annotation performed by

SnpEff and most related software packages will broadly categorize missense substitutions and their accompanying amino acid changes (e.g. K154-¿L154) as moderate impact variants. Short indels whose length is a multiple of three are treated similarly, unless they introduce a stop codon, as their effect will usually be localized.

Once missense and frame-preserving indel variants are identified, a more detailed estimation of their impact on protein function can be performed using heuristic and statistical models. The most common approaches are based on conservation, either amongst orthologous or homologous proteins, or protein domains, sometimes adding information of the physio-chemical properties of the reference and variant amino acids (e.g. differences in side chain charge, hydrophobicity, or size). The SIFT algorithm27 assesses the degree of selection against specific amino acid changes at a given position of a protein sequence by analyzing the substitution process at that site throughout a collection of predicted homologous proteins identified by PSI-BLAST28. Based on this multiple sequence alignment and the highly conserved regions it contains, SIFT calculates a normalized probability of amino acid replacement (called the SIFT score), which estimates the mutations effect on protein function.Polyphen29, another commonly used tool, takes the process one step further by searching UniProtKB/Swiss-Prot30 and the DSSP database of secondary structure assignments31 to determine if the variant is located in a known active site in the protein.In contrast to other methods that categorize each variant individually, VAAST32, a commercially available package, computes scores for groups of variants located within a given gene and "collapses" them into a single category, a concept similar to burden testing performed for rare variants identified in exome sequencing studies.For human proteins, SnpEff makes use of the Database for Nonsynonymous SNVs Functional Predictions33 (dbNSFP), which collects scores produced by several impact assessment algorithms in a single database. Individually, impact assessment methods usually have an estimated accuracy of 60% to 80%, but predictions from several algorithms can be combined to provide a stringent, but more accurate estimate of impact35.

In most cases these algorithms apply best to SNVs since these are common in populations and there is more genomic sequence andexperimental data available to refine the statistical methods. However, some recently developed algorithms are capable of assessing variants other than SNVs, including PROVEAN34, which extends SIFT to assess the functional impact of indels.

Caveats

Imprecise models of protein function. Accurate impact assessment of coding variants remains an open problem and most computational predictions are riddled with both false positives and false negatives. While both missense variants and frame-preserving indels are broadly cataloged as having moderate effects, this is mostly due to lack of a comprehensive model and the extremely complex computations that would be required for an in-depth analysis (such as protein structure predictions). In these cases, proteomic information can be revealing. SnpEff adds annotations from curated proteomic databases, such as NextProt 36, which can help to elucidate if a mutation alters a critical protein amino acid or domain (such as amino acids that are post-translationally modified as part of a signaling cascade or that are form the active site of an enzyme) resulting in a protein may no longer function.

Gain of deleterious function. Computational variant annotation may eventually be able to fairly accurately predict the molecular impact of a variant in terms of the degree to which it translates in a loss of function for the encoded protein. However, gains of function, including the acquired ability to interact with new partners and disrupt their function, remain vastly more difficult to tackle, although a several such variants have been linked to disease, such as hereditary pancreatitis 37.

Unanticipated effects of synonymous variants. In most cases, synonymous variants are regarded as non-deleterious (or low impact); however, one needs to seriously consider the possibility that they may have greater functional effects by altering mRNA splicing 38 or secondary structure 39. Synonymous SNVs may also alter translation efficiency, by changing a frequently used to a rarely used codon and have been linked to changes in protein expression 40.

3.4.5 Non-coding variant annotation

Although coding variants represent less than 2% of variants in the human genome, they make up the vast majority of confirmed disease-related variants that have been validated at a functional level. This may result from ascertainment bias (since variants in coding regions are straightforward to discover and characterize at a basic level and many studies have largely ignored non-coding variants); or may be explained by the increased complexity of computational approaches and lab assays required to predict and validate the impact of non-coding variants; or by their potentially more subtle impact on gene expression or cell function. Nonetheless, in a compendium of current GWAS studies, roughly 40% of the variants are intergenic and 30% intronic and functional studies of these variants are increasingly emphasizing the importance of non-coding genetic variation at risk loci for complex genetic diseases and traits41.

Functional non-coding regions of the genome encompass a wide variety of regulatory elements contained in DNA and RNA molecules that are involved in transcriptional and post-transcriptional regulation. Cis-regulatory elements include (i) binding sites for DNA-binding proteins such as transcription factors and chromatin remodelers; (ii) binding sites for RNA-binding proteins involved in splicing, mRNA localization, or translational regulation; (iii) micro RNA (miRNA) target sites; and (iv) long non-coding RNA (lncRNA) targets on DNA, RNA and proteins. Non-coding transcripts include well-characterized regulatory RNAs (e.g. miRNA, snoRNA, snRNA, piRNA and lncRNAs) as well as RNAs involved directly in protein synthesis (e.g. tRNA and rRNA). The annotation and impact assessment of non-coding variants presents a significant challenge for several reasons: (i) reliable technologies to study transcriptional regulatory regions on a genome-wide basis are only just reaching maturity and provide limited resolution of binding sites for individual transcription factors and regulatory RNA molecules; (ii) non-coding functional regions of most genomes remain incompletely mapped as they vary widely among different cell types and cell states (for example, in diseased and healthy tissues); (iii) non-coding regulatory elements often are part of complex transcriptional programs that are time-dependent, contain many redundant linkages or reciprocal connections between genes and respond to a wide range of intra- and extracellular signals; and (iv) genomic regulatory elements rarely have a strict consensus sequence (for example, compare the position weight matrices used to identify transcription factor or miRNA binding sites with the amino acid triplet code) making the effect of a mutation on gene regulatory programs difficult to predict. As a result, high-quality annotation of non-coding variants relies more heavily on experimental data than is the case for coding variants: since many of these experimental techniques did not study the effects of SNVs on gene regulatory programs, they can only be used to annotate variants and not to predict their effects on gene transcription. In the few cases where the effects of SNVs have been studied (for example, the effects of SNVs that are common in a population and located in genetic loci associated with complex diseases), experimental approaches provide highly accurate functional assessment at a cost of reduced coverage compared to computational approaches.

Large-scale projects such as ENCODE7and modENCODE42 have made major steps toward mapping gene transcription and transcriptional regulatory regions in many tissues and cell types, but similar studies in diseased tissues remain at an early stage (for example, the growing collection of disease-related epigenomes from the Epigenome Roadmap8). The base-by-base resolution and number of cell states studied for different types of regulatory elements and non-coding transcripts varies widely among datasets; in part due to the lack of sensitive, comprehensive and high-resolution technologies to study the different molecular species and modes of interaction that can be altered by non-coding variants. Efficient technologies for genome-wide, high-throughput mapping of binding sites for RNA-binding proteins (PAR-CLiP43), miRNAs (PAR-CLiP44 and CLASH45) are starting to be applied on a broad scale as are protocols to map transcription factor binding sites (TFBS) which can improve resolution to a single base (Chip-exo46). However, in most cases, DNA and RNA binding sites are only imprecisely located within Chip-Seq peaks that span genomic regions hundreds of base pairs in length, with computational approaches being used to pinpoint the bases most likely mediating the interaction. In the absence of more precise localization data,de novo computational prediction of binding sites for DNA and RNA binding proteins remains insufficiently accurate to be of much use in annotating single noncoding variants.

This limitation is particularly critical for functional predictions of putative target sites for microRNAs and other regulatory RNA species. MicroRNAs are short RNA molecules that regulate gene expression post-transcriptionally by binding the messenger RNA of a gene through complementary, usually in the 3 region of the transcript, which leads to mRNA degradation or inhibits translation. Sequence variants that cause the loss or gain of a miRNA target site would lead to dysregulation of the gene, with likely deleterious effects. Although miRNAs are relatively well documented in most model organisms including human, their binding sites are only starting to be mapped experimentally, and computational predictions has very low specificity. Meaningful information regarding the possible role of a variant in disrupting a miRNA target site is starting to emerge47, although variants that create new miRNA binding sites remain under the radar.

Even if the position of a functional element could be perfectly determined, predicting a variants impact on chromatin conformation, promoter activity, gene expression, or transcript processing remains challenging. For transcription factors, this involves predicting whether the protein will still be able to recognize its mutated site (and with what affinity), as well as predicting the impact of these changes on gene expression levels. The latter is particularly hard to predict as a result of interactions, competition, and redundancy contained in regulatory networks of transcription factors or RNA binding proteins. As a consequence, computational prediction of the functional impact of non-coding variants remains a very active area of research and there is no broad consensus on the best methodology to use48. One significant exception is the identification of variants affecting canonical splice sites, defined as two bases on the 3 end on the intron (splice site acceptor) and 5 end of the intron (splice site donor). Variants that affect canonical splice sites are easily detected and typically lead to abnormal mRNA processing, involving exon loss or extension that leads to loss of function of the encoded protein.

3.4.6 Impact assessment of non-coding variants

Two broad classes of publicly available genome-wide datasets are commonly combined to assess the functional impact of non-coding genetic variants: (i) computational predictions of sequence conservation and sites involved in molecular interactions such as transcription factor and RBP binding, as well as miRNA-mRNA target interactions; and (ii) experimental genome-wide localization assays for DNA binding proteins, histone modifications, and chromatin accessibility.

Computational sources of evidence:. Interspecies sequence conservation plays a key role in scoring and prioritizing non-coding variants. This is based on the assumption is that sites or regions that have been more conserved across species than expected under a neutral model of evolution are likely to be functional; suggesting that mutations contained in them are likely to be deleterious. In the absence of strong experimental data, sequence conservation measures calculated from whole genome multiple alignments, (for example using PhastCons 49, SciPhy 50, PhyloP 51 , and GERP 52), have been developed to provide a generic indicator of function for non-coding variants. Although high conservation scores generally mean that a genomic region may be functional, the converse is not true and many experimentally proven functional noncoding regions show only modest sequence conservation (for example due to binding site turnover events). Finally, some regulatory regions (e.g. specific elements regulating immune response 53) are under positive selection and may thus show less conservation than surrounding neutral regions.

In human, genome-wide computational predictions of transcription factor binding sites based on matching to publically available position weight matrices are available from variety of sources, including Ensembl 5 and Jaspar 54. Because of the low information content of most binding affinity profiles, the specificity of the predictions is generally very low. Related approaches exist to predict splicing regulatory regions 55 and miRNA target sites 47,56, some of which are precomputed for whole genomes and available from the UCSC or Ensembl genome browsers. Recent efforts to determine RNA-binding protein sequence affinities can also be used to identify putative binding sites for these proteins in mRNA 57.

Experimental sources of evidence:. To investigate the potential impact of variants on transcriptional regulation, many published experimental data sets produced by large-scale projects such as ENCODE 7, modENCODE 42 and Roadmap Epigenomics 8, can be used directly by annotation packages. These include: (i) ChIP-seq or ChIP-exo experiments that identify TFBSs on a genome-wide basis; (ii) DNAseI hypersensitivity or Formaldehyde-Assisted Isolation of Regulatory Elements (FAIRE) assays that identify regions with open chromatin; and (iii) ChIP-seq studies to identify the presence of specific promoter or enhancer-associated histone post-translational modifications, which can be combined to identify active, poised, and inactive enhancers and promoters 57. Most of these data sets are easily available through Galaxy 3 (as tracks from the UCSC Genome Browser) or through SnpEff (as downloadable pre-computed datasets). In parallel with the types of studies described above, expression quantitative trait loci (eQTLs) represent an agnostic way to map putative regulatory regions. An increasing number of such loci are available through the GTEX database 58. Experimental data that may support assessment of the impact of variants on post-transcriptional regulation remain sparser, although databases such as doRiNa 59 or starBase 60 contain genome-wide datasets obtained by CLIP-Seq and degradome sequencing. To our knowledge, these data have yet to be used in the context of variant annotation studies.

Combining sources of evidence:. Despite the variety of computational and experimental sources of evidence available, impact assessment for non-coding variants remains relatively crude, due to the fact that biological models of gene regulation remain fairly simple. Nonetheless, significant steps forward have been made recently, and two web-based tools, HaploReg 61 and RegulomeDb 62, perform SNV and indel impact assessment for variants from dbSNV on the basis of a broad body of computational and experimental evidence. Both use pre-computed scores for variants from dbSnp and therefore cannot be used for rare variants, but they are extremely valuable for exploration by associating the variant of interest with a variant in dbSnp via linkage disequilibrium.

Caveats

Sparseness of functional sites within ChIP-seq peaks. Even if a non-coding variant is located in a region that contains a ChIP-seq peak for a given TF and has all the hallmark signatures of regulatory chromatin, the likelihood that it is deleterious remains low, because most DNA bases contained within a peak are non-functional.

Gain of function mutations. While this section, has focused on variants causing the loss of a functional regulatory element, genetic variants may also create new or more effective transcription factor binding sites. These are substantially harder to detect as they can occur in regions that show no evidence of function in individuals possessing the reference allele, and show little conservation across species. Furthermore, computational methods to predict gain of affinity for a given TF caused by a variant have insufficient specificity to be of much use on their own.

3.5 Clinical effect of variants

One of the most revealing types of annotation of both coding and noncoding variants reports whether the variant has previously been implicated in a phenotype or disease. Although such information is available for only a small minority of all deleterious variants, their number is growing and should be the first type of annotation one seeks out. Clinical annotations, until recently, have been scattered in a large number of specialized databases of medical conditions with a genetic basis, including the comprehensive, manually curated collection of genetic loci, variants and phenotypes in the Online Mendelian Inheritance in Man database (OMIM, http://www.omim.org); web pages containing detailed clinical and genetic information about uncommon disorders in the Swedish National Board of Health and Welfare Database for Rare Diseases (http://www.socialstyrelsen.se/rarediseases) and the peer-reviewed NIH GeneReviews collection (54://www.ncbi.nlm.nih.gov/books/NBK1116); and a curated collection of over 140,000 mutations associated with common and rare genetic disorders in the commercial Human Gene Mutation Database (HGMD, http://www.hgmd.org/). In most cases, these datasets do not use standardized data collection or reporting formats; are designed to primarily provide information to patients and health professionals through a web interface; and rely on heterogeneous criteria to describe disease phenotypes and clinical outcomes; pathological and other clinical laboratory data; as well as the genetic and biologic experiments that have been used to demonstrate disease mechanisms at a molecular or cellular level. These shortcomings are being addressed by initiatives that provide centralized, evidence-based, comprehensive collections of known relationships between human genetic variants and their phenotype that are suitable for computational analysis, such as the

NIH effort to aggregate records from OMIM, GeneReviews and locus-specific databases in ClinVar (http://www.ncbi.nlm.nih.gov/clinvar).

Another important application of variant detection and annotation is in the study of cancer genomes, which is occurring increasingly in clinical settings to support treatment decisions for advanced tumors. Annotation of variants detected in tumor sequences can be analyzed for clinical cohorts, using similar techniques as other complex traits, as well as for individual patients, using techniques to identify differences between somatic (tumor) and germline (healthy) tissues. In the latter case, one looks for cancer-associated mutations that distinguish the somatic genome of cancer cells of an individual from the germline genome in order to find the driving mutations that pinpoint the specific mechanisms underlying tumorigenesis or metastasis. Ideally, these mutations can be used to select a treatment for the patient, establish prognosis, or to identify causative mutations that have led to the cancers progression. In such a setting, given that sequence differences between the cancer and germline genomes are of greater interest than the background genetic changes between the germline and a reference genome, variant calling is performed using specialized algorithms, such as MuTect 63 and SomaticSniper 64.

Once variants are called, variant annotation focuses on somatic variants that are not present in the germline genome, which is the new "reference genome". Although SnpEff was originally developed for the study of germline genomes, it also contains modules that allow the annotation of cancer mutations. A seemingly simple approach would be to create a new reference genome using the individuals germline genome, and then annotate somatic mutations by comparison to this new reference. Unfortunately, this approach would be laborious and computationally expensive, so a preferred solution is to compare each genome to the reference human genome, and reconcile shared differences by creating a germline genome "on the fly" only for those regions that require it (i.e. variants that are shared by the germline and cancer genome are disregarded). This optimization reduces the processing time from hours to only a few seconds, making it viable for analysis of hundreds of samples simultaneously.

Caveats

Annotation accuracy. Biological knowledge, as well as molecular and phenotypic evidence supports the identification of certain groups of high impact variants based on simple criteria (such as premature stops, frameshifts, start lost and rare amino acid mutations); however, it is often hard to predict whether non-synonymous variants will have equally large effects on an organism's health. Even when the accepted "rules of thumb" used in the primary annotation indicate that protein function is impaired, we should consider that these predictions may be based on a small number of model genes and will require appropriate wet-lab validation or confirmatory studies in cohorts. In addition, as more human genomes are sequenced, it is likely that some genetic variants that have been linked to Mendelian diseases will be found in healthy individuals 65; and in many cases, may not actually be disease-causing mutations 66.

BOX 1: Data structures and computational efficiency

Most of the computational pipelines for genomic variant annotation and primary impact assessment are relatively efficient and can annotate variants obtained from large resequencing projects involving thousands of samples within a few minutes or hours even using a moderately powered laptop. This is typically achieved through two key optimizations: (i) creation of reference annotation databases and (ii) implementation of efficient search algorithms. Reference database creation refers to the process of creating and storing precomputed genomic data from the reference genome, which can be searched quickly to extract information relevant to each variant. This process needs to be performed only once per reference genome and most annotation tools have pre-computed databases for many organisms available for users to download (for instance, SnpEff currently offers databases for over 25,000 organisms). Since these databases are typically quite large, efficient search algorithms are used together with appropriate data structures to optimize the search process. In ANNOVAR 9, each chromosome is subdivided in a set of intervals of size k and genomic features for a given chromosome are stored in a hash table of size L/k, where L is the length of the chromosome. Another approach, used by SnpEff, is to use an "interval forest", which is a hash of interval trees 68 indexed by chromosome. Querying an interval tree requires O[log(n) + m] time, where n is the number of features in the tree and m is the number of features in the result. Both approaches are extremely efficient.

BOX 2: Reference genomes and gene annotations

A reference genome is the standard against which every genome within a species is compared. For example, the latest assembly of the human reference genome from NCBI is hg19, which is equivalent to ENSEMBLs GRCh37. A reference genome is not static; and new and improved assemblies are released frequently for newly assembled genomes and less frequently for more mature genomes such as human or mouse. Because genomic coordinates may change from one assembly to the next, it is critical that all analyses of genomic variants in a given project be done with respect to the same version of the reference genome (ideally the most recent), and also that the version of reference genome used in an analysis be reported in publications. Mapping reads to one reference genome, while performing variant annotations using another reference genome is a common mistake that leads to disastrous results.

Gene annotations consist of the genomic coordinates of every known isoform for all genes in the genome. This includes the position of the start and end of transcription, splice sites, and in the case of protein-coding genes, the start and stop codons. In contrast to genome assemblies, there are typically several different sources of gene annotations for each genome; and these sources are often not consistent with each other. For example, human genome annotations include the RefSeq genes 69, a set of well characterized and highly curated genes, as well as the UCSC 70 and ENSEMBL 71 gene annotations, which have higher coverage but are less stringently curated. Gene annotation revisions are frequent and may be made as part of a specifically identified "genome version" or on a continuing basis. Whereas ENSEMBL releases new sub-versions of their genome annotation several times a year (e.g. GRCh37.72 is human reference version 37, subversion 72), this is not the case for most other genome annotation providers, where changes in genome annotation happen asynchronously and unannounced. For those, it is important to report the full transcript ID (e.g. NC_XXXX.S), which will identify the annotation subversion, because future transcript subversions might correct genomic annotation errors. In most cases, coordinates from different genome annotation versions based on the same reference genome assembly will be the same, but it is not uncommon to find a gene in a different position, or even a different chromosome.

Despite the huge amount of work being done to curate gene annotations in reference genomes, some errors and oddities persist; including non-phased start positions, proteins without start or stop codons, incomplete transcripts, 4-base codons, 1-base introns, etc. Indeed, the predicted amino acid sequences for about 10% of the transcripts in the RefSeq database do not exactly match the corresponding protein. These cases may be due to errors in the reference genome assembly or genome annotation, but may also reflect rare post-transcriptional regulatory events such as RNA editing. Because they confuse most variant annotation pipelines, these cases can easily lead to an overestimation of the potential impact of a coding variant. SnpEff can identify the most common of these sources of errors and flag suspicious cases for the user to review.

BOX 3: Approaches to standardization

Bioinformatic standards make it possible to create programs that interoperate and share complex datasets. In the absence of a “one size fits all” format to describe genes, proteins and genetic variants, different file formats are used for different purposes, each one having their strengths and weaknesses. A crucial part of bioinformatics analysis is to use the right file format appropriately, so here we introduce file formats and standards used most commonly used variant annotations. Many bioinformatics formats are text based and can be read using a text editor or as a spreadsheet, which are convenient ways to identify problems or debug analysis strategies.

VCF (Variant Call Format): This format, introduced by the 1000 Genomes project, provides a standard for describing genetic variants. Each line in a VCF file (record) represents a genomic location (a variant) and is described by metadata in "fields" separated by tabs. A VCF file record contains eight mandatory fields: i) chromosome name (CHROM), ii) position (POS), iii) variant name (ID), iv) reference allele (REF), v) alternative allele (ALT), vi) variant call quality which is an error probability estimation (QUAL), vii) filter pass or filter fail parameters (FILTER), and viii) a generic container for information (INFO). The INFO field is used to add additional metadata in a semi-structured way and is the field where annotations can be added. Here is an example of a few lines of a VCF file:

```
#CHROM POS     ID        REF    ALT     QUAL FILTER INFO
20     14370   rs6054257 G      A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2
20     17330   .         T      A       3    q10    NS=3;DP=11;AF=0.017
20     1110696 rs6040355 A      G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB
20     1230237 .         T      .       47   PASS   NS=3;DP=13;AA=T
20     1234567 microsat1 GTC    G,GTCT  50   PASS   NS=3;DP=9;AA=G
```

Missing data is indicated by a period. The mandatory fields may be followed by optional genotype fields - one column per sample sequenced or genotyped - that describe whether the genetic variant was observed in that sample.

In the VCF standard, the REF field (column 4) identifies the base present in the reference genome and is independent of the samples studied in a particular experiment. This is an important distinction for cancer samples, where it is common to compare somatic to germline sequences using a "virtual reference genome" (the germline genome). For compatibility with other software that uses VCF files, this "virtual reference" should not be placed in the REF field. Somatic and germline samples are treated just as two samples added as two genotype fields (to columns after column 9).

BED (Browser Extensible Data) Files: This simple format is often used to describe intervals or regions in the genome. Annotation software allows users define custom intervals using BED formatted files, since they can be easily created and manipulated in spreadsheets or text documents. The BED format recognizes one interval per line: the line can include metadata separated by spaces or tabs that describes: i) the chromosome name, ii) the start position as a zero-based coordinate, iii) the end position as a one-based coordinate. These mandatory fields can be followed by other labels or scores (see https://genome.ucsc.edu/FAQ/FAQformat.html#format1 for details). The unusual coordinate choice of zero- / one-based in the start and end position is a source of many headaches and plenty of confusion amongst researchers, but provides more flexibility in describing genomic regions. For example, it is possible to define a zero length interval by providing the same coordinate in both fields (which is not possible in many other formats).

HGVS nomenclature: A standard used to describe variants occurring in a protein, DNA or RNA molecules, which has emerged as a standard in translational research. As a simple example of this format "p.Arg22Ser" describes a variant changing amino acid number 22 from Arginine to Serine. The standard is quickly evolving in an attempt to make it comprehensive for all types of variants observed in sequencing studies.

Variant annotation and sequence ontology. Recent efforts have been made to standardize the output format of variant annotation tools. The Human Genome Variation Society (HGVS) created a format to describe protein, RNA and DNA mutations that has emerged as a standard in translation research. As a simple example of this format "p.Arg22Ser" describes a variant that changes amino acid number 22 from Arginine to Serine. This format is supported by most annotation software packages, at least to some degree. Another format gaining momentum is the "Sequence Ontology", which is an ontology of sequence changes and their putative effects (e.g. "stop_gained" or "missense", see Table 1). These two standards help to provide a unified vocabulary for annotations that helps to avoid nomenclature problems and artificial barriers between different software programs. Hopefully, standards for annotations can propagate into the VCF specification, as this will make it easier to transfer datasets between different annotation and analysis packages, change processing pipelines, or even use multiple packages providing complementary functionality. All these tasks are now difficult due to lack of standards for describing the results of large-scale genotyping and next generation sequencing studies.

Appendix A

A.1 Algorithm details

Here we show the details on the algorithmic implementations...

References

[1] C.T. Abdallah. Mathematical controllability of genomic networks. *Proceedings of the National Academy of Sciences*, 108(42):17243–17244, 2011.

[2] N.J. Cowan, E.J. Chastain, D.A. Vilhena, J.S. Freudenberg, and C.T. Bergstrom. Nodal dynamics, not degree distributions, determine the structural controllability of complex networks. *Arxiv preprint arXiv:1106.2573*, 2011.

[3] A.P. Fejes, G. Robertson, M. Bilenky, R. Varhol, M. Bainbridge, and S.J.M. Jones. FindPeaks 3.1: a tool for identifying areas of enrichment from massively parallel short-read sequencing technology. *Bioinformatics*, 24(15):1729, 2008.

[4] M.J. Fullwood and Y. Ruan. ChIP-based methods for the identification of long-range chromatin interactions. *Journal of cellular biochemistry*, 107(1):30–39, 2009.

[5] D. Ghosh and Z.S. Qin. Statistical Issues in the Analysis of ChIP-Seq and RNA-Seq Data. *Genes*, 1(2):317–334, 2010.

[6] M. Hu, J. Yu, J.M.G. Taylor, A.M. Chinnaiyan, and Z.S. Qin. On the detection and refinement of transcription factor binding sites using ChIP-Seq data. *Nucleic acids research*, 38(7):2154, 2010.

[7] H. Ji, H. Jiang, W. Ma, D.S. Johnson, R.M. Myers, and W.H. Wong. An integrated software system for analyzing ChIP-chip and ChIP-seq data. *Nature Biotechnology*, 26(11):1293–1300, 2008.

[8] S. Jiao, C.P. Bailey, S. Zhang, and I. Ladunga. Probabilistic Peak Calling and Controlling False Discovery Rate Estimations in Transcription Factor Binding Site Mapping from ChIP-seq. *Methods in molecular biology (Clifton, NJ)*, 674:161–177, 2010.

[9] R. Jothi, S. Cuddapah, A. Barski, K. Cui, and K. Zhao. Genome-wide identification of in vivo protein-DNA binding sites from ChIP-Seq data. *Nucleic acids research*, 36(16):5221, 2008.

[10] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[11] H. Li. Mathematical Notes on SAMtools Algorithms. *http://lh3lh3.users.sourceforge.net/download/samtools.pdf*, 2010.

[12] H. Li and R. Durbin. Fast and accurate short-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(5), 2009.

[13] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589, 2010.

[14] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078, 2009.

[15] F. Lin. Robust control design an optimal control approach. 2007.

[16] Y.Y. Liu, J.J. Slotine, and A.L. Barabási. Controllability of complex networks. *Nature*, 473(7346):167–173, 2011.

[17] P.J. Park. ChIP–seq: advantages and challenges of a maturing technology. *Nature Reviews Genetics*, 10(10):669–680, 2009.

[18] S. Pepke, B. Wold, and A. Mortazavi. Computation for ChIP-seq and RNA-seq studies. *Nature methods*, 6:S22–S32, 2009.

[19] W. Song, W. Jianmin, Z. Wei, P. Stanley, and C. Cheng. ChIP-PaM: an algorithm to identify protein-DNA interaction using ChIP-Seq data. *Theoretical Biology and Medical Modelling*, 7.

[20] A.M. Szalkowski and C.D. Schmid. Rapid innovation in ChIP-seq peak-calling algorithms is outdistancing benchmarking efforts. *Briefings in Bioinformatics*, 2010.

[21] C.M. Taniguchi, B. Emanuelli, and C.R. Kahn. Critical nodes in signalling pathways: insights into insulin action. *Nature Reviews Molecular Cell Biology*, 7(2):85–96, 2006.

[22] L. Teemu, R. Sunil, T. Soile, L. Riitta, and A. Tero. A practical comparison of methods for detecting transcription factor binding sites in ChIP-seq experiments. *BMC Genomics*, 10.

[23] Stanford University. Wnt home — stanford university, nussel lab.

[24] Wikipedia. Wnt signaling pathway — Wikipedia, the free encyclopedia. 2004.

[25] E.G. Wilbanks and M.T. Facciotti. Evaluation of algorithm performance in ChIP-seq peak detection. *PloS one*, 5(7):e11471, 2010.

[26] Y. Zhang, T. Liu, C.A. Meyer, J. Eeckhoute, D.S. Johnson, B.E. Bernstein, C. Nussbaum, R.M. Myers, M. Brown, W. Li, et al. Model-based analysis of ChIP-Seq (MACS). *Genome biology*, 9(9):R137, 2008.