UNIVERSITA' DEGLI STUDI DI VERONA

DIPARTIMENTO DI

_____*Informatica*_____

SCUOLA DI DOTTORATO DI

_____*Scienze naturali ed ingegneristiche*_____

DOTTORATO DI RICERCA IN

_____*Informatica*_____

Con il contributo di (ENTE FINANZIATORE)

_____

CICLO /ANNO (1° anno d'Iscrizione) _____34_____

TITOLO DELLA TESI DI DOTTORATO

Personal genome editing algorithms to identify increased variant-induced off-target potential

S.S.D. ___INF/01_____
(indicare il settore scientifico disciplinare di riferimento della tesi dato obbligatorio)*

Coordinatore: Prof./ssa _Massimo Merro_____

Firma _____

Tutor: Prof./ssa __Nicola Bombieri_____

Firma _____

Dottorando: Dott./ssa Samuele Cancellieri

Firma _____

Quest'opera è stata rilasciata con licenza Creative Commons Attribuzione – non commerciale
Non opere derivate 3.0 Italia . Per leggere una copia della licenza visita il sito web:

http://creativecommons.org/licenses/by-nc-nd/3.0/it/

Attribuzione Devi riconoscere una menzione di paternità adeguata, fornire un link alla licenza e indicare se sono state effettuate delle modifiche. Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.

NonCommerciale Non puoi usare il materiale per scopi commerciali.

Non opere derivate —Se remixi, trasformi il materiale o ti basi su di esso, non puoi distribuire il materiale così modificato.

Personal genome editing algorithms to identify increased variant-induced off-target potential

Samuele Cancellieri
Tesi di Dottorato
Verona, 10 Dicembre 2021
ISBN 12324-5678-910

Sommario

Abstract

Clustered regularly interspaced short palindromic repeats (CRISPR) technologies allow for facile genomic modification in a site-specific manner. A key step in this process is the in-silico design of single guide RNAs (sgRNAs) to efficiently and specifically target a site of interest. To this end, it is necessary to enumerate all potential off-target sites within a given genome that could be inadvertently altered by nuclease-mediated cleavage.

Off-target sites are quasi-complementary regions of the genome in which the specified sgRNA can bind, even without a perfect complementary nucleotides sequence. This problem is known as, off-target sites enumeration, and became common after discovery of CRISPR technology.

To solve this problem, many in-silico solutions were proposed in the last years but, currently available software for this task are limited by computational efficiency, variant support, genetic annotation, assessment of the functional impact of potential off-target effects at population and individual level, and a user-friendly graphical interface designed to be usable by non-informatician without any programming knowledge.

This thesis addresses all these topics by proposing two software to directly answer the off-target enumeration problem and perform all the related analysis. In details, the thesis proposes CRISPRitz, a tool designed and developed to compute fast and exhaustive searches on reference and alternative genome to enumerate all the possible off-target for a user-defined set of sgRNAs with specific thresholds of mismatches (non-complementary bps in RNA-DNA binding) and bulges (bubbles that alters the physical structure of RNA and DNA limiting the binding activity).

The thesis also proposes CRISPRme, a tool developed starting from CRISPRitz, which answers the requests of professionals and technicians to implement a comprehensive and easy to use interface to perform off-target enumeration, analysis and assessment, with graphical reports, a graphical interface and the capability of performing real-time query on the resulting data to extract desired targets, with a focus on individual and personalized genome analysis.

1 Introduction

Clustered regularly interspaced short palindromic repeats (CRISPR) genome editing has revolutionized the ability to modify a genome of interest in a targeted and programable way (Cong et al., 2013; Mali et al., 2013). The initially described CRISPR system for eukaryotic genome editing involves a single guide RNA (hereafter referred to as a guide or sgRNA) to direct Streptococcus pyogenes-derived Cas9 (SpCas9) protein for site-specific genomic cleavage upstream of a protospacer adjacent motif (PAM), which is NGG for SpCas9. The Cas9-mediated double strand break is repaired by endogenous repair pathways including non-homologous end joining (NHEJ), microhomology-mediated end joining (MMEJ) and homology-directed repair (HDR). NHEJ/MMEJ often result in the introduction of insertions/deletions (indels) while exploitation of the HDR pathway allows for precise integration of customized sequence by providing a donor repair template (Komor et al., 2017). Since the initial description of eukaryotic genome editing by SpCas9, the CRISPR toolbox has been greatly expanded to include a variety of novel- and modified-nucleases with distinct PAM sequences (e.g. Cas12a, Cas9 derived from different species, and modified-Cas nucleases) (Komor et al., 2017). Although designed for site-specific cleavage, CRISPR nuclease-mediated cleavage may occur at other genomic sites, termed off-target sites. Off-target cleavage(s) commonly occur at sites of sequence homology to the on-target site; however, the rules governing off-target cleavage are incompletely understood. In general, mismatches may lead to a reduction in cleavage activity or have no effect at all depending on the specific base change and the relative position as described in (Doench et al., 2014, 2016a)

2 Background

This chapter gives a simple introduction to the main concepts necessary to understand the work. Starting from the biology of CRISPR/Cas protein to the algorithm and methods for in-silico off-targets enumeration and assessment.

Section 2.1 is a general background about CRISPR/Cas, why it's a revolutionary technology and how it works, explaining also some of the problems involved in the technology itself.

Section 2.2 is a general background about in-silico analysis of CRISPR/Cas off-targets, state-of-the-art software and how it's so important.

Section 2.3 explains the current status of the field and concentrate more on why personal genetics and software capable of using variants will be fundamental in the immediate future.

2.1 CRISPR/Cas

CRISPR/Cas is a genetic engineering technique used for DNA cutting. The name of this method comes from the main components used by it: CRISPR and Cas. Those two components allow to manipulate the DNA through recognition of a precise target sequence on the target DNA.

Firstly, we will introduce the natural CRISPR/Cas working. Afterward, we will explain the differences between natural and engineered CRISPR/Cas. Finally, we will illustrate the off-target problem related to the use of engineered CRISPR/Cas for DNA cutting.

2.1.1 Natural CRISPR/Cas

CRISPR (Clustered Regularly Interspaced Short Palindromic Repeats) is a group of DNA sequences composed by the alternation between two different types of strings (see Figure 1):

Protospacer is a peculiar large sequence motif presents in the DNA. A motif is a pattern very common in the genomes, which has a biological meaning. Often, it indicates sequence-specific binding sites for proteins such as nucleases (D'haeseleer, 2006). In the CRISPR sequence there are many different protospacers, which are involved during the system formation and each of them is used by a CRISPR/Cas complex to match with the target region.

SPR (Short Palindromic Repeats) is a short-repeated DNA sequence, in which each repetition is interspersed by a protospacer. The SPR length is less than that of a protospacer. In the complex, SPR cooperates with the tracrRNA sequence (we will illustrate it later) to support the protospacer to detect the region where a CRISPR/Cas complex will bind.

Protospacers and SPRs are the fundamental part of the CRISPR system, which is used by the bacteria to protect themselves from external agents, such as viruses or plasmids (Wiedenheft et al., 2012). CRISPR system is able to identify the presence of an external factor because protospacer, and a part of SPR, may match with the target sequences.

CRISPR system requires to build the complexes beginning from proto-spacers and SPR to make its identification and destruction operations. First, CRISPR sequence is transcribed by the enzyme RNA polymerase into in an RNA sequence called pre-crRNA (see Figure 1). After the CRISPR transcription, there are three factors that bind with the pre-crRNA:

- RNAase III is a specific nuclease that catalyzes an RNA string into a smaller sequence. Nucleases like RNAase III are called ribonucleases, because they work on the RNA (Ribonucleic Acid). More precisely, nucleases have a classification based on their activity: endonucleases and exonucleases, where the first group of enzymes acts in the inner part of sequences. On the contrary, the second group digest nucleotides starting from the sequence ends. Therefore, ribonucleases have the same first classification where RNAase III belongs to the endoribonucleases. RNAase III is able to recognize and cleave a dsRNA (double strand RNA) sequence (Nicholson, 2014) in order to separate the pre-crRNA in the right position.
- Cas (CRISPR associated) is the fundamental DNA endonuclease enzyme associated with the bacteria immunity system; the most common used Cas belongs to Streptococcus pyogenes. In this organism are present 93 Cas genes that generate several Cas proteins. Cas genes are grouped following the sequence similarity of the produced enzymes. The most used Cas enzyme is Cas9 (CRISPR associated protein 9, also called SpCas9) and it is composed by only one bilobed protein. It is widely used because it produces a double-strand breaks and, separately, it may produce a single-strand breaks and it is able to use many different RNA sequences to recognize distinct target strings. Therefore, Cas9 is used within the complex to cut the double-strand in the target detected site.
- tracrRNA (trans-activing crRNA) is a small RNA sequence encoded upstream of the Cas operon (Karvelis et al., 2013). Operon is a cluster of genes that encodes the pre-crRNA and Cas9 enzyme. TracrRNA is involved during crRNA maturation, and it is useful to recognize the SPR present in the pre-crRNA.

The above three elements bound with the pre-crRNA and they make the crRNA:tracrRNA:Cas9:RNAase III (see Figure 1). The complex is made for each pair of protospacer and SPR present on the pre-crRNA. In the next step RNAase III cut the pre-crRNA sequence in order that each pair of protospacer and SPR splits from the other pairs of sequences, consequently RNAase III leaves the final complex. Accordingly, we will have several cr-RNA:tracrRNA:Cas9 complexes (see Figure 1). Concluded this phase, each complex is ready to begin its function.

The CRISPR/Cas system is able to detect the target sequence present on the double-strand DNA thanks to its nucleic acid sequence composed by SPR and protospacer cut from the crRNA. More in detail, part of SPR recognizes the PAM (Protospacer Adjacent Motif) present near the target sequence, instead, the target sequence is matched by the protospacer because it is complementary to the protospacer (see Figure 1).

After the binding between CRISPR/Cas complex and the target sequence, the double-stranded DNA is opened by helicase that breaks the hydrogen bonds between the nucleotide bases presents on the opponent DNA strands and the Cas9 may implement its endonuclease activity to cut both strands of the DNA (see Figure 1). Finally, the crRNA:tracrRNA:Cas9 complex leaves the double-stranded DNA cut (see Figure 1).

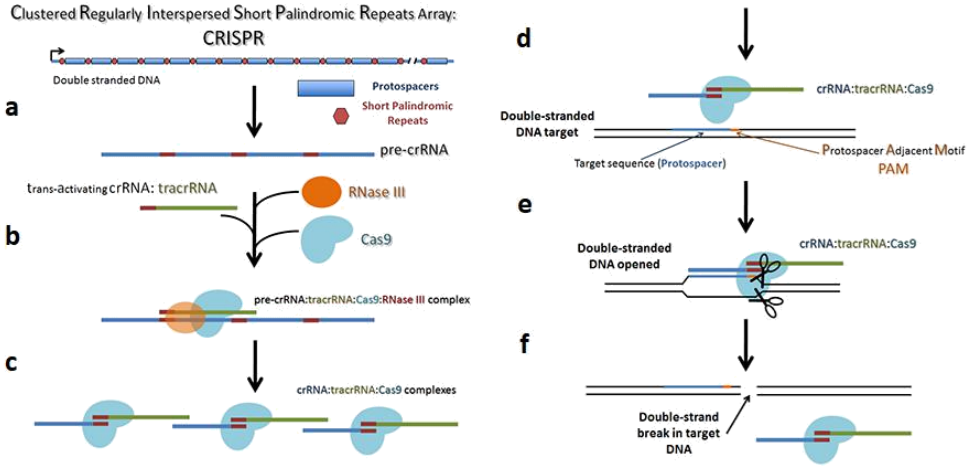


Figure 1. Naturally work of the CRISPR/Cas system.

2.1.2 Engineered CRISPR/Cas

The CRISPR/Cas system is a good and efficient machine, which is able to recognize a specific target sequence on the DNA. Therefore, CRISPR/Cas complex has been adapted to make a genome editing technology that allows to manipulate DNA through strength cuts (Ran et al., 2013). The adaptation of the natural complex is called engineered CRISPR/Cas. The reason why it is preferable to use an engineered CRISPR/Cas with respect to a natural system is that the used RNA for the first complex is simpler to build preserving the same experiment results (Jinek et al., 2012). Moreover, the RNA sequence is programmable, then it is an efficient tool used in different cells and organisms. The basic components of the engineered CRISPR/Cas system are different from the natural complex components: the Cas9 endonuclease is equal in both complexes, on the contrary, the RNA used for the target identification is called guide RNA (gRNA) and it is composed by the union between a part of crRNA and a specific trascrRNA (Sander & Joung, 2014). Then, the engineered complex does not require to bind together crRNA and trascrRNA to build the complex, because crRNA and trascrRNA sequences are also fused in a unique string during the transcription. The guide RNA folds creating a linker loop (see Figure 2) and it assumes a right conformation to be able to recognize the target sequence on the DNA using twenty or few more nucleotides at the 5' end, which corresponds to a portion of protospacer of the crRNA in the natural complex (see Figure 2). The engineered system may use a guide RNA of length twenty or more nucleotides because it has been seen that protospacer is able to match with a different target sequence simply by altering the first twenty nucleotides of the guide RNA. Instead, the PAM is recognized by the trascrRNA sequence.

The guide RNA is now ready to bind with the Cas enzyme and CRISPR/Cas complex may recognize the target portion of the DNA sequence in the same way of the natural complex.

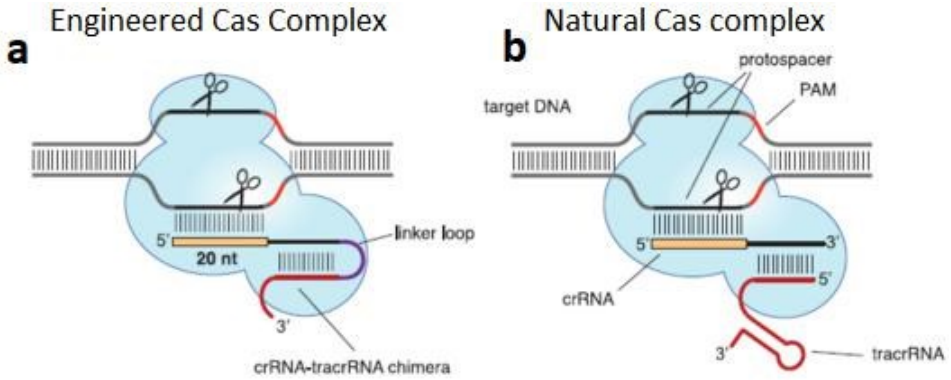


Figure 2. Naturally and engineered CRISPR/Cas complex

2.1.3 Off-Targets problem

CRISPR/Cas procedure is widely used in DNA manipulation. However, it is well known that CRISPR/Cas9 genome editing may cleave undesirable target sites not fully complementary to the guide RNA. Therefore, there is an important question to analyze when we want to use the CRISPR/Cas system: the guide RNA specificity. Several studies proposed some solution to solve the guide RNA specifity problems called also "off-target", for instance, lowering Cas9 expression or truncating the guide RNA sequences used for targeting at the 5'-end.

However, such options may be inefficient because they may decrease on-target cleavage percentage (Slaymaker et al., 2016) and it is not yet clear the guide RNA tolerance during the match between itself and the target region. Matching errors are generally better tolerated at the twenty nucleotides that bounds with the target region. On the contrary, several PAM are able to recognize more different sequences. However, it does not allow to have matching errors during the bond with a target sequence.

The quality results given by CRISPR/Cas system works are mainly influenced by the used guide RNA, because it is the main component responsible for recognizing the target regions of the DNA that will be to cut. When a CRISPR/Cas complex binds to a target sequence it is possible to obtain two different results:

- On-target is obtained when the complex binds to an expected target sequence on the DNA, completely complementary to the guide RNA (see Figure 2)
- Off-target is obtained when the system recognizes a not complementary target sequence as complementary of the guide RNA and the complex binds to it. The identification of a wrong target region may be caused by three possible events.
    - Mismatch is when one or more (generally not more than four) nucleotides match without following the Watson-Crick rules (FEUGHELMAN et al., 1955) (see Figure 3).
    - RNA bulge is a particular behavior of the RNA guide where the strand folds with one or more (generally not more than two) nucleotides (see Figure 3).
    - DNA bulge is similar to RNA bulge, but in this case, is the DNA sequence that folds (see Figure 3)

The reason why off-target problem is so significant is that off-target, means that the nucleases will cut the genome in positions that can compromise experiment results and can also have potential implications for medical uses of the CRISPR/Cas technology (Kleinstiver et al., 2016).

Therefore, several papers have tried to define some constraints able to make a biological preview about where off-target sites will appearing. For instance, it is observed that there is no correlation between the total number of off-target sites detected by a guide RNA and the GC content of the on-target protospacer. Furthermore, off-target are founded dispersed through the genome in exons, introns and noncoding intergenic regions (Tsai et al., 2015). With the actual knowledge, we do not have a reliable set of rules about this phenomenon, there exists only isolated tools that may preview where and how many off-target sites it is possible to obtain. Nevertheless, it is not sure which off-target sites a guide RNA will really bind during the CRISPR/Cas system processing, if not before testing it in a laboratory of molecular biology.

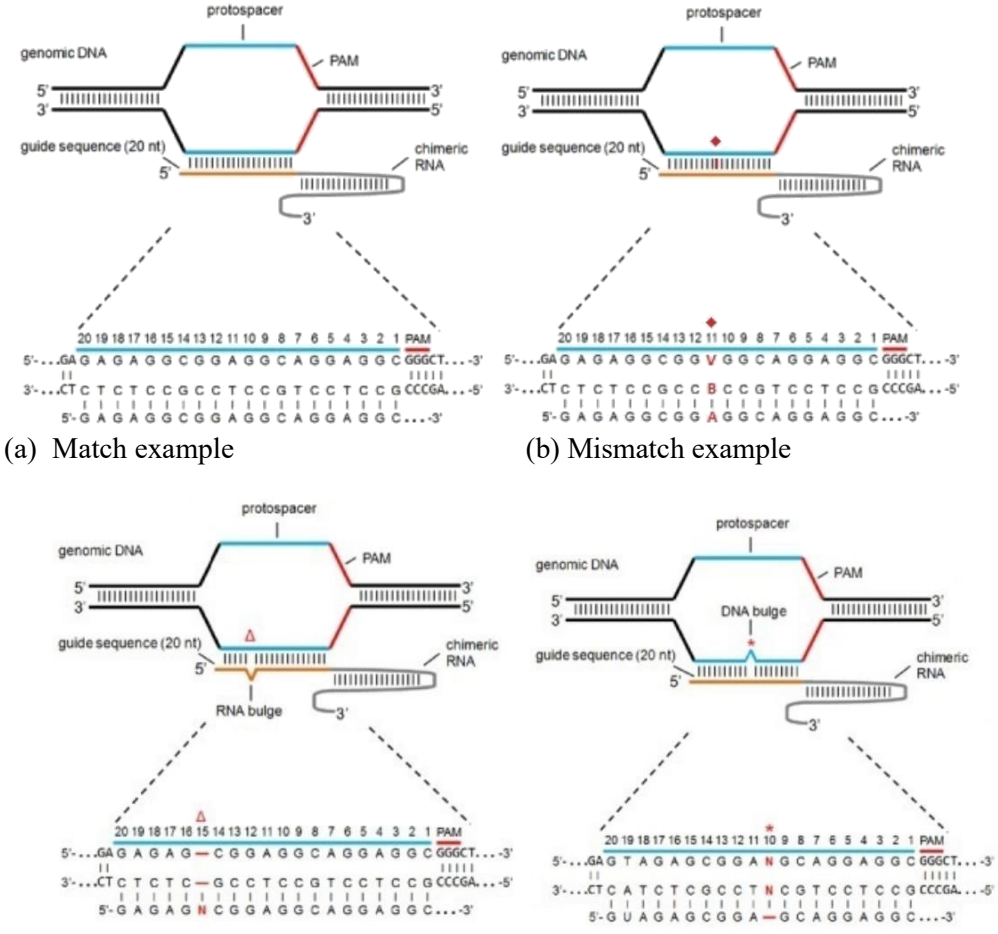


Figure 3. Examples of on-target (a) and off-target (b, c, d)

2.2 Computational off-target enumeration and state-of-the-art

CRISPR/Cas9 is a powerful system that enables researchers to manipulate the genome of target cells like never.

If we want to do scientific studies by using CRISPR/Cas9, we should follow some different steps to perform our study. First, we must select the desired kind of genetic manipulation, because each specific manipulation requires different CRISPR components. Moreover, there may not be a perfect plasmid for our specific application, in such a case, it may be necessary to customize an existing reagent. Secondly, it is important to select the expression system, which will depend upon the specific application. Finally, we choose the target sequence, and we design an appropriate guide RNA useful to find the target string. During the last step we select one or more guide RNA based on predicted on-target and off-target activity.

In this chapter, we desire to focus on the tools that allow to obtain useful information about on-target and off-target activity. To obtain the desired knowledge we may use software based on two different approaches: a guide RNA design program or an off-target site prediction tool.

Different software for the design of a single guide RNA or a pool of guide RNAs have been developed and it are capable of locating potential PAM and target sequences and ranking the associated guide RNAs based on their predicted on-target and off-target activity. Off-target site prediction tools are, instead, dedicated, to check a guides pool, in fact, the user gives the RNA guide(s), the maximum number of allowed mismatches and the genome of interest, afterwards, the software analyses the performance of the given RNA guide(s) and returns the possible off-target sites found in the input genome. Therefore, it does not generate a guide RNA database starting from a chosen genome because it is necessary to know the guide RNA before launching the software.

When studies on the use of the CRISPR/Cas technique to manipulate the genome began, off-target site prediction tools or guide RNA design program not existed yet. The first algorithm adapted for the CRISPR/Cas behavior prediction was Burrows-Wheeler Aligner (BWA) (Li & Durbin, 2009). The reason why BWA algorithm was used is that off-target activity prediction may be considered as a problem of string alignment of a word (guide RNA) on a text (genome) in which BWA is very efficient. Now, the situation is different, because there are many tools specialized on the off-target prediction or

on the guide RNA design, even if some of these software are based on BWA algorithm, for instance CRISPOR (Haeussler et al., 2016).

During the study of the current situation about CRISPR/Cas tools, we put more attention on these algorithms:

- GuideScan (Perez et al., 2017) is an efficient algorithm of guide RNA design, which is able to create a single or a pool of guide RNA starting from the targeted genome.
- BWA (Li & Durbin, 2009) is the first algorithm used for off-target site prediction, even if it was not invented exactly to solve the off-target problem.
- CRISPOR (Haeussler et al., 2016) is an algorithm of guide RNA design, which is based on BWA, able to make also off-target site prediction starting from more designed guide RNAs obtained from a given DNA sequence.
- Cas-OFFinder (Bae et al., 2014) is a fast and versatile software for potential off-target site prediction, which is usable also on GPUs. Moreover, it has many add-ons useful for a researcher to perform more in-depth analysis.
- crispRtool (Lessard et al., 2017) is an R script used to process variant genome and found invidual off-targets. It is not a stand-alone tool since it was used in Lessard

(Lessard et al., 2017) to perform a specific analysis with 1000 Genome Project VCFs.

In the next sections, we will explain the software listed above, explaining how they work and motivating why we decided to use Cas-OFFinder as main competitor for our analysis.

2.2.1 GuideScan

GuideScan is an example of design software, which allows to build a completely customizable guide RNA database. The power of GuideScan is to design guide RNAs that are more specific than those designed by other tools. GuideScan works following three steps (see Figure 4):

1. The user supplies the targetable genome as FASTA file. Moreover, GuideScan allows the user to provide additional information by defining the value of three parameters:
    - Canonical PAM, through the definition of the desired Cas enzyme. For instance, if the user wants to use the NGG PAM, he or she will select Cas9. The user may also specify non-canonical PAMs, since they may be recognized by the system and contribute to off-target cutting.
    - PAM position relative to the guide RNA binding sequence.
    - Guide RNA length, in some cases truncating a guide RNA into a sequence shorter than 20 nucleotides complementary of a target region, it is possible to improve the used guide RNA specificity without sacrificing on-target genome editing efficiency (Fu et al., 2014).

Furthermore, the user supplies two different Hamming distances (called also mismatches): "M" for which guide RNAs are required to have a unique target site in the genome and "Q" for which potential off-target sites will be enumerated. Therefore, GuideScan examines the input le to identify all k-mers associated with canonical and non-canonical PAM.

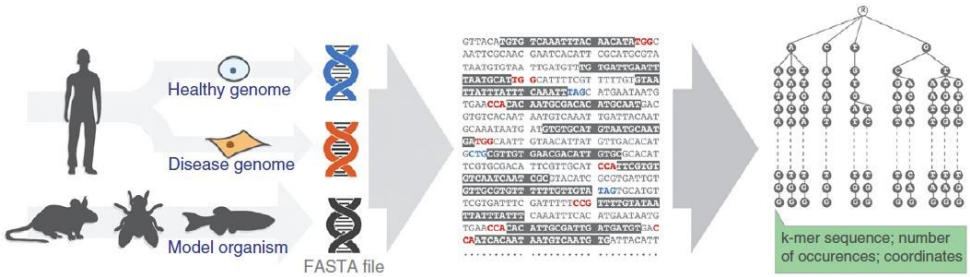


Figure 4. GuideScan workflow: left, input FASTA file of chosen genome; middle, canonical (red) and non-canonical (blue) PAM define targetable sequences; right, candidate guide RNAs are indexed in the trie (Bentley & Sedgewick, 1998a).

1. GuideScan enumerates all targetable sequences present in the genome, making a list of all k-mers and determinates which of these are potential guide RNAs. Then, it constructs a retrieval tree (trie) data structure of all k-mers:
    - If a k-mer occurs with a canonical PAM uniquely in the genome, it is labelled as a candidate guide RNA.
    - On the contrary, if the k-mer occurs more than once or with a non-canonical PAM, it is labelled as a non-candidate guide RNA. All guide RNAs relabeled as non-

candidates are written into the blacklist, which contains all k-mers rejected as candidate guide RNA.

GuideScan ensures a unique target in the genome up to "M" mismatches for each candidate guide RNA, having a distinction between two candidates guide RNA by at least "M" mismatches. The software guarantees the explained condition through a trie traversal, in which it evaluates a mismatch neighborhood for each candidate guide RNA:

- If there are mismatches between candidate guide RNA and one of its neighbors, then the candidate guide RNA is not unique in the genome up to "M" mismatches, it is relabeled non-candidate and written into the blacklist.
- Otherwise, a candidate guide RNA may have off-target sites completely enumerated (where Q > M) up to "Q" mismatches.

GuideScan does again a trie traversal and computing the mismatch neighborhood for off-target information of a given guide RNA. Therefore, thanks to the trie structure, we are sure that until "M" mismatches between the target sequence and the guide RNA there are no o-target and for more than "Q" mismatches there are only off-target.

- Finally, all candidate guide RNAs are retrieved from the trie and saved in a SAM file. Each guide RNA sequence is stored as a unique identifier in the file, and it also stores information about off-target sites for each guide RNA. Then, the SAM file is converted in a BAM file.

2.2.2 Burrows-Wheeler Aligner

Burrows-Wheeler Alignment tool (BWA) is an implementation of a read alignment software based on backward search with Burrows Wheeler Transform (BWT) (Li & Durbin, 2009). BWA is very efficiently at aligning short and long sequencing reads against a large reference sequence such as the human genome. BWA is an inexact string-matching algorithm, because it allows to have mismatches and gaps between strings.

Before to explain the BWA workflow we need to introduce some basic information useful to know all the aspects of the tool:

- The prefix trie (Bemer, 1961; Fredkin, 1960) for string *X* is a tree where each edge is labelled with a character of the string *X* and the resulting string form the concatenation of the edge characters on the path from a leaf to the root is a unique prefix of the string *X*. Each node of the trie represents the substring of the string *X* given by the resulting string concatenation of the edge characters starting from considered node to the root. The resulting string associated to the node is a unique substring of the string *X*. It is important to note that the prefix trie of the string *X* is identical to the suffix trie of reverse of string *X*, suffix trie theories then may also be applied to prefix trie.
- The suffix array (SA) (Manber & Myers, 1993) for a string *X* of length *n* is an array of integers of range *1 to n* specifying the lexicographic order of the suffixes of the string *X*. It will be convenient to assume that *X[n] = $*, where *$* is smaller than any other character (see Figure 5). It is possible to calculate a suffix array having a prefix trie.
- The Burrows-Wheeler Transform (BWT) (Li & Durbin, 2009) is a rearrangement of a string. BWT is used for compression, because it is a reversible transformation and it is easier to compress string with many repeated characters. BWT of a string *X*, which has an additional last character *$* lexicographically smaller than all other characters, is defined as a string *BWT* where *BWT[i]=$* when the suffix array SA, which is calculated through

the prefix trie, has *SA(i)=0* and *BWT[i]=X [SA(i)-1]* otherwise. It is important note that the string *X*, the string *BWT* and the suffix array *SA* have all the same length.

BWA is a recursive algorithm to search for the suffix array intervals of substrings of the string *X* (input string, then the genome) that match with another shorter string *W* (query, then the guide RNA). It is possible to launch BWA to perform an exact or inexact matching. During the matching between the query *W* against the string *X* it is allowed to have no more than *n* differences (mismatches or gaps) between them.

BWA uses backward search (a technique that match the query *W* on the string *X* starting from the last character of the query) to sample distinct substrings from the genome. Therefore, it needs a pre-calculation before starting the real matching.

During the pre-calculation BWA uses BWT algorithm to create the rearranged string *B* calculated on the input string *X*. Afterwards, the algorithm calculates the array *C*, where *C(a)* is the number of symbols of the string *X*, that are lexicographically smaller than a, and the array *O*, where *O(a,i)* is the number of occurrences of *a* in *B[0,i]*. Therefore, BWA calculates the rearranged string *B'* for the reverse reference of the input string *X* and the corresponding *O'*. Note that BWA does not need to calculate again *C'* because the symbols remain the same, but they are only permuted.

Afterwards, there is the effective procedure where BWA searches the query *W* (guide RNA) into the string *X* (genome). The "inexact search" is done in two steps:

- CalculateD procedure uses the BWT of the reverse reference sequence to test if a substring of the query *W* is also a substring of the string *X* and writes the array *D*, where *D(i)* is the lower bound of the number of differences in *W [0,i]*. Note that during the CalculateD step, BWT uses both *B* and *B'* in order to have an *O(jW j)* procedure rather than $O(|W|^2)$.
- InexRecur recursively calculates the *SA* intervals of substrings that match *W* with no more than *n* differences. InexRecur descends into the subtrees in the prex trie of the query *W* to calculate if the substrings, which are found in the CalculateD step, have more than *n* mismatches.

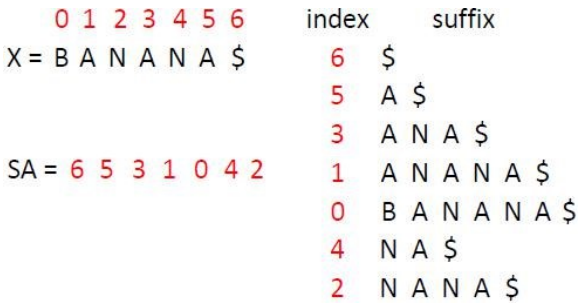


Figure 5. Example of Suffix Array.

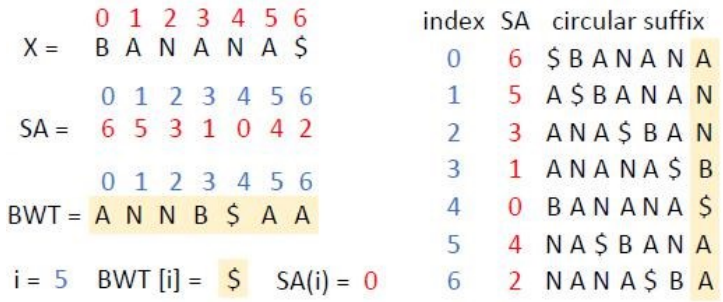


Figure 6. Example using string "BANANA".

2.2.3 CRISPOR

CRISPOR is a software based on BWA, which helps to select and express guide RNA sequences. CRISPOR supports many genomes, but they have to be pre-computed before using.

CRISPOR improves the results before to give them to the user. The genomic hits retrieved from BWA are filtered for the requested PAM sequence and scored. Therefore, they are annotated with gene model information using the UCSC Genome Browser command line tools (Kent et al., 2002).

Below, we will explain each step of CRISPOR:

- First, CRISPOR requires three parameters from the user:
  - Input Sequence: the user specifies an input DNA sequence (no special sequence format is required) where he wants to find the guide RNA. Characters different from A, C, G, T and N (Adenine, Cytosine, Guanine, Thymine and any nucleotide respectively) will be automatically removed from the input sequence by the software. It may not be submitted RNA sequences; indeed U (Uracil) would be removed. In case of the input DNA sequence contains some N characters, no guides will match with these characters. Therefore, the user may use the N character to mark positions that wants to exclude from the design, for instance, to avoid SNPs (single nucleotide polymorphisms) (Kreitman, 1983)(Fujimori et al., 1989). The input DNA sequence should usually be contained in the selected genome, but it may not be present. For instance, during the design of some guide RNAs against a trans-gene, such as GFP (Green Fluorescent Protein) (Auer et al., 2014).
  - Genome: the user selects the genome of interest from a list of 113 pre-calculated genomes. The default genome automatically selected is the Homo Sapiens - Human (hg19/GRCh37 (Browser, 2013)). For some species there are multiple different versions of the genome, for instance Homo Sapiens - Human and Mus Musculus - Mouse, because sequences of different years may have different loci. More-over, the annotation with variants (SNPs and short indels) is only available for certain genome version, for instance, the 1000 Genomes variant annotation ("A Map of Human Genome Variation from Population-Scale Sequencing," 2010) is only available for the human genome assembly called hg19/GrCh37.
  - PAM: for the most current applications of the CRISPR/Cas system is used the Streptococcus pyogenes Cas9 endoribonuclease, which the corresponding PAM is NGG. Cas9 is the default PAM used by

CRISPOR, however, the user may choose other enzymes and corresponding PAMs from those available on CRISPOR.

- CRISPOR finds the possible guide RNAs in the input DNA sequence through the matching between the PAM associated with selected Cas and the input DNA sequence. Therefore, CRISPOR matches the guide RNAs found with the chosen genome by using BWA aligner. Each possible guide RNA taken from the input DNA sequence is aligned against the whole genome allowing at most four mismatches. The results then are summarized in a table.

- The output of CRISPOR is composed by two parts:

    - In the first part of the results there is the annotated input DNA sequence in which all PAMs sites are highlighted. PAM sites may also be annotated over the reverse strand of the input sequence, showed by the reverse complement of the PAM. For instance, if the chosen PAM is SpCas9 (NGG) and it matches with the reverse strand of the input DNA sequence, then the shown PAM over the input DNA sequence will be "CCN".

- If the genome contains annotated variants, CRISPOR shows the annotated variants (mostly SNPs) associated with the input DNA sequence.

    - With the input DNA sequence, it is possible to view additional information taken from UCSC or Ensembl, depending on the source of the genome.
    - In the second part of the output, results are illustrated in a summary table, which is sorted by the Specificity score. We will clarify the content of each column:
        i. Guide name is composed by the position of the PAM on the input DNA sequence and the strand where it matches (reverse or forward).
- Guide sequence is the sequence taken from the input DNA sequence. The taken guide RNA will depend on the selected PAM, for instance, the spCas9 PAM is NGG and it targets sequences 20bp (base pair) long. In Guide sequence column there are also the PAM and the link to its PCR and cloning primers, which is the feature of CRISPOR that allows to retrieve many extra information about the guide RNA. In addition, depending on the genome and guide RNA, additional data are displayed such as variants if they are available for this genome.
        ii. Specificity score is a column where the score is a prediction of how much a guide RNA sequence, for the target site, may lead to off-target cleavage somewhere else in the genome. Specificity score and the next scores have range from 0-100 where 100 is the best. In Specificity score, the best meaning that the research could not find a single sequence in the genome, which differs from the target sequence until four mismatches.
        iii. Efficiency scores is a prediction of how well the target site may be cut by the guide RNA sequence.
        iv. Out-of-frame score is a prediction of how likely a guide is to lead to out-of-frame deletions. It is relevant if the user is doing gene knockouts with a single guide RNA (Shalem et al., 2014). Gene knockouts with single guide RNA works because repair after

DNA cutting is error-prone and small deletions are introduced 5' of the PAM.

v. Off-target mismatch counts is the number of possible off-targets in the genome for each number of mismatches. It is a summary of the whole-genome research for sequences similar to the guide RNA. The total number of off-targets is shown in this column, too.

vi. Off-targets is the column where CRISPOR lists the locations of all possible off-targets with up to four mismatches, annotated with additional information: genomic position and an annotation whether they fall into an exon, intron or between genes and the closest gene.

CRISPOR Batch is another way to use CRISPOR, which is available for users who want to use pre-selected guide RNAs for gene inactivation experiments in mouse (Kühn et al., 1995) or human cells (Herman et al., 1995). It accepts one or multiple genes, which are identified by Entrez Gene IDs or Refseq IDs, and returns several pre-selected guide RNAs from various genome-wide libraries.

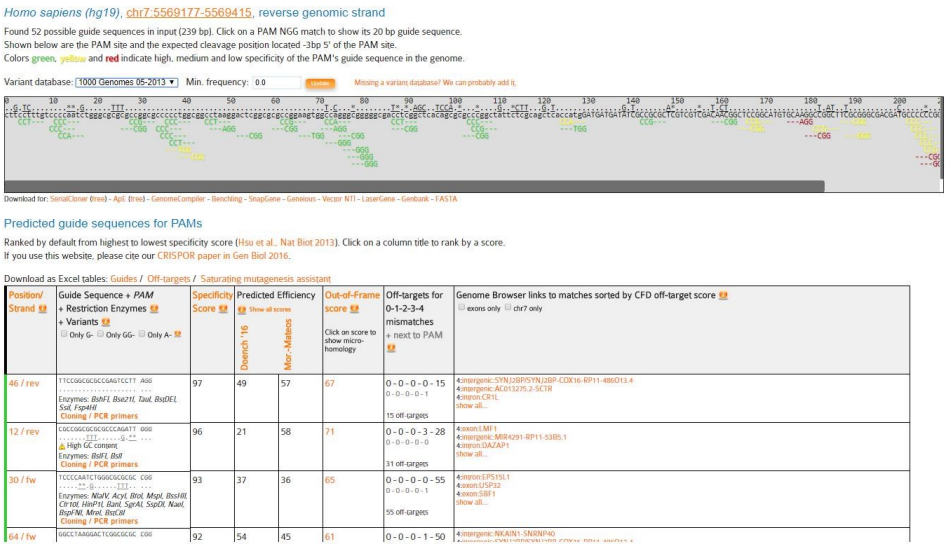


Figure 7. Example of CRISPOR results.

2.2.4 Cas-OFFinder

Cas-OFFinder (Bae et al., 2014) is a fast and highly versatile off-target searching tool useful to preview the off-target activity of a guide RNA. Cas-OFFinder is written in OpenCL, which is an open standard language for parallel programming based on C++ executable in diverse platforms such as central processing units (CPUs) and graphics processing units (GPUs). We choose to analyze it because it is widely used, and it has one of the most used guide RNA design tools.

When a researcher uses Cas-OFFinder, he or she chooses the desired Cas, the genome to be analyzed, the input guide RNA and the off-target constraints (which corresponds to the maximum number of allowed mismatches, RNA bulges and DNA bulges) (see Figure 8).

Cas-OFFinder uses the input information to make the following research (see Figure 9):

- Searching the pattern in the chosen genome, which corresponds to the sequence of the chosen PAM, and saves the position indices found during the searching.
- Comparing the guide RNA inserted by the user with the target sequence found in the genome during the PAM searching.

After illustrating briefly, the concept of on-line version of Cas-OFFinder, we want to illustrate more in detail the workflow of its off-line version. The functioning of the two versions is the same, the only difference is that in the online version the user does not

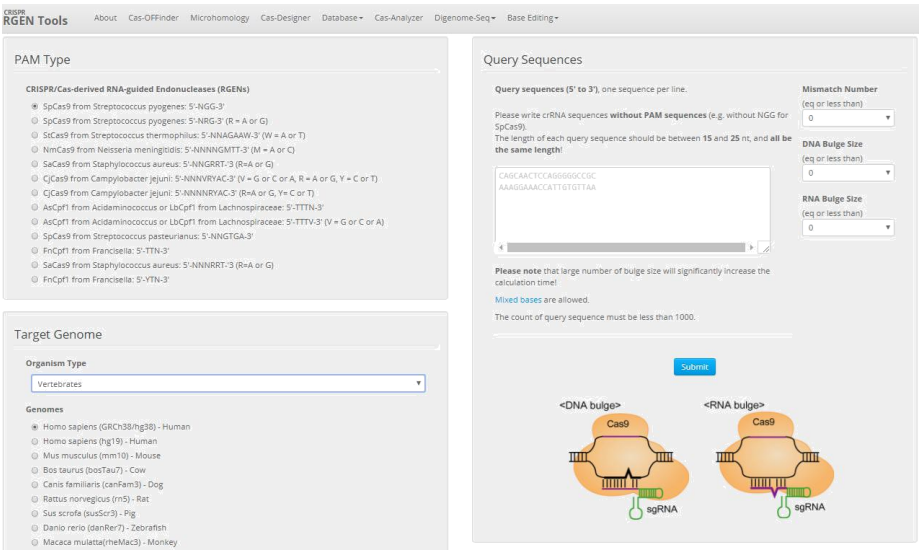


Figure 8. Input of Cas-OFFinder (website version).

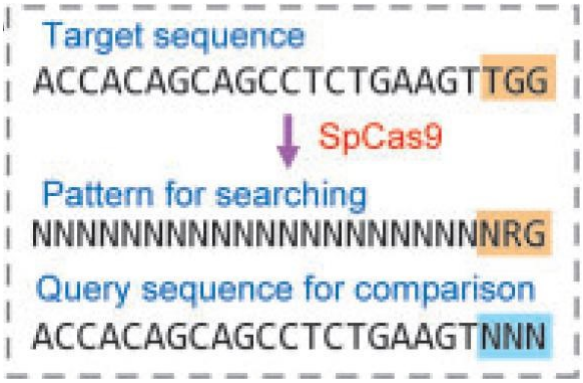


Figure 9. Process of Cas-OFFinder.

provide the file of the genome because it is already present in the server of Cas-OFFinder. The software is composed by two different OpenCL kernels (a searching kernel and a comparing kernel) and three C++ wrappers (see Figure 10):

- Cas-OFFinder reads genome sequence data files in single or multi-sequence FASTA formats. It is possible to provide the genome also in two-bit version (compressed version of a FASTA file).
- Wrapper1 divides the genomes in the largest possible sequences (called chunk) where the size is the maximum allowed by the device memory (CPU or GPU). Wrapper1 is fundamental for big data analysis, because the memory of the device could be not always large enough.
- Searching kernel loads the divided chunks and compiles all sites that include the PAM sequence. It searches and selects these specific sites quickly because Searching kernel runs independently on every calculation unit processor.
- Wrapper2 collects the information about the specific sites containing the PAM sequences.
- Comparing kernel receives the selected sequences collected by the Wrapper2. It matches the selected sequences with the input guide RNA and count the number of mismatched bases. Comparing kernel works similarly as Searching kernel does.
- Wrapper3 selects potential off-target sites that have fewer mismatched bases than the user defined. Therefore, it writes the following information into an output file: chromosome number, off-target DNA sequence, off-target position in the genome, direction (strand) and number of mismatched bases.
- The software repeats the previous steps until it has analyzed the whole input genome.

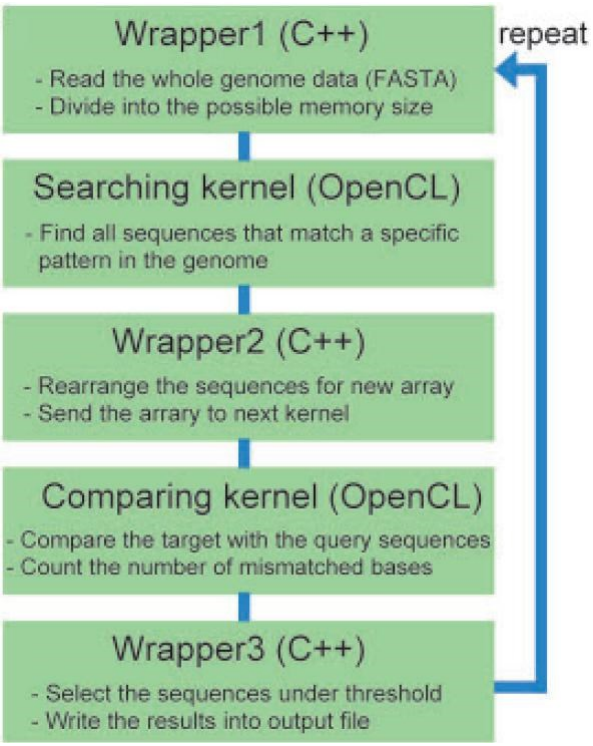


Figure 10. Workflow of Cas-OFFinder.

We choose Cas-OFFinder as main competitor, which will be compared with the software proposed by this thesis. We decided to not utilize the other software and start from Cas-OFFinder to develop our algorithm for the following different reasons:

- We sought a simple and fast software usable with every personal computer.
- GuideScan and CRISPOR website versions were very powerful, fast and produced much information, but the off-line versions did not guarantee to obtain the same results in a short time.
- GuideScan and CRISPOR were guide RNA design program, but our idea was to make an off-target prediction tool.
- The real power of Cas-OFFinder was in the simplicity of the basic version that it was extended in case of necessity.
- BWA was a very good candidate as a competitor, but it needed too many changes in the source code to adapt it to our needs. For instance, the number of off-target sites found by BWA are far fewer than those found by Cas-OFFinder using the same initial setup.

2.2.5 crispRtool

crispRtool was used in Lessard (Lessard et al., 2017) to perform specific analysis on reference and alternative genome, using 1000 Genome Project VCFs. This study was performed to prove the importance and relevance of variants in the discovery of new putative off-targets.

The tool, consists in a simple R script, that takes as input:

- FASTA file containing single-guide RNA(s)
- PAM sequence (only SpCas9)
- VCF file (optional)
- Max number of allowed mismatches

The tool searches the input sgRNA on the default genome (hg38), returning a list of putative off-targets with different scores (Doench et al., 2016b; Ran et al., 2013; Sanjana et al., 2014).

If a set of VCFs data is provided, the script converts any alternative nucleotide using the IUPAC ambiguity code (Johnson, 2010) and process each off-target by returning a subset of valid off-targets containing alternative alleles.

Output of the script will consist in two separate files:

- .matches.txt, containing information on the genomic matches for each guides. It contains the following columns:
- sgRNA ID: sgRNA ID
  a. sgRNA Seq: sgRNA sequence
  b. chr: Chromosome of match
  c. start: Start position of match
  d. end: End position of match
  e. strand: Strand of match
  f. match Seq: Sequence of match
  g. score: Individual score of match
  h. CFD: CFD Score
  i. mismatches: Number of mismatches
- .summary.txt, containing information on the total number of matches and overall score of each guides. It contains the following columns:
- sgRNA ID: sgRNA ID
  j. sgRNA Seq: sgRNA sequence
  k. nontargeting score: Non-targeting guide score. This score assumes that the sgRNA should NOT match any sequence of the genome.
  l. best match position: Position of best match
  m. best match score: Score of best match
  n. best match CFD: CFD Score of best match
  o. targeting guide score: Targeting guide score. This score assumes that the sgRNA should have one genomic match that is the intended target. If there is no perfect match, this score will be equal to the non-targeting score.
  p. mean CFD: Mean CFD Score
  q. median CFD: Median CFD Score
  r. max CFD: Maximum CFD Score
  s. min CFD: Minimum CFD Score
  t. sd CFD: Standard deviation of CFD Score
  u. perc CFD: 10,25,75, and 90th percentile of CFD score.
  v. Total matches: Total number of matches
  w. N X: These columns contains the number of matches with X mismatches. There will be X+1 columns ranging from N 0 (perfect match) to N X, where X is the maximum number of allowed mismatches

The script, proved to be the only competitor capable of work with a public available dataset (1000 Genomes Project Consortium & others, 2015) of consistent size (over a hundred million of variants in total) reporting a comprehensive list of off-targets with variants. This script was used as comparison for the second software proposed in this thesis, CRISPRme, since it was the only one capable of performing the aforementioned search.

2.3 Personal genetics in CRISPR/Cas off-targets enumeration and analysis

Personal genetics is a very discussed and new topic in biotechnology and medicine. The possibility to personalize a treatment to maximize the effect and avoid any unwanted side effect will revolutionize any medical field.

CRISPR/Cas experiments and following treatments, aim exactly to this target. So, in the last years, many new trial and tests were performed adapting each treatment to a subset of individuals to test the outcomes and how these treatments will adapt to different people and conditions.

Obviously, these results are yet to be common or perfect and many more steps must be taken to obtain secure and sound treatments.

Currently, the problem of working with genetical variants, it's limited by the lack of a satisfying coverage, there are few available comprehensive and accessible datasets and the lack of efficient and powerful tools, either for wet-lab experiments and in-silico prediction.

Regarding the genetic dataset availability, one of the well-known and most used datasets, is the dataset of 1000 Genome Project (1000G from now on) (1000 Genomes Project Consortium & others, 2015), containing more than 2500 individuals selected from 5 different geographical areas, fully sequenced with phased genotyping.

Another well-known and comprehensive dataset is represented by the Human Genome Diversity Project (HGDP from now on) (Foster, 2008), containing more than 900 individuals from 7 different global areas, sequenced with unphased genotyping.

But other than these datasets, there is a lack of comprehensive and accessible data, since many project are inaccessible for research purpose or they lack the coverage of different worlds populations.

The situation from a computational perspective, is similar. Many software are available to do off-target enumeration and prediction, but very few are variant-aware and nearly anyone can complete a genome-wide search with a huge dataset as the 1000 Genome Project dataset in acceptable time.

This situation originates from the problems and the difficulties that working with variant data. These are some of the issues that needed to be addressed in order to use variant data in this kind of analysis. The first issue that needed to be solved, it's the capability to work with many alternative alleles without making the problem exploding in terms of computational time. When working with these huge datasets, like 1000G, we must work with many individual presents in the analysis, and we need to understand exactly how each individual's variants are changing the outcomes of the analysis. For example, the 1000G dataset, contains 2504 individuals from different populations and it's necessary to treat each individual as a separate person from any other. So, it's necessary to develop a solution to work with these data, without generating a personal genome for each individual and then perform searches on all these genomes.

After solving this problem, the data needs to be processed to understand how each alternative target is affecting the results, for example introducing a critical mutation in a dangerous genomic region, so each target needs to be analyzed and any information needs to be extracted and explained to the user. This process needs to be carefully designed and

developed since the volume of output data is huge and any error in the design phase can cost a huge increase in the computational and execution time in post-processing phases.

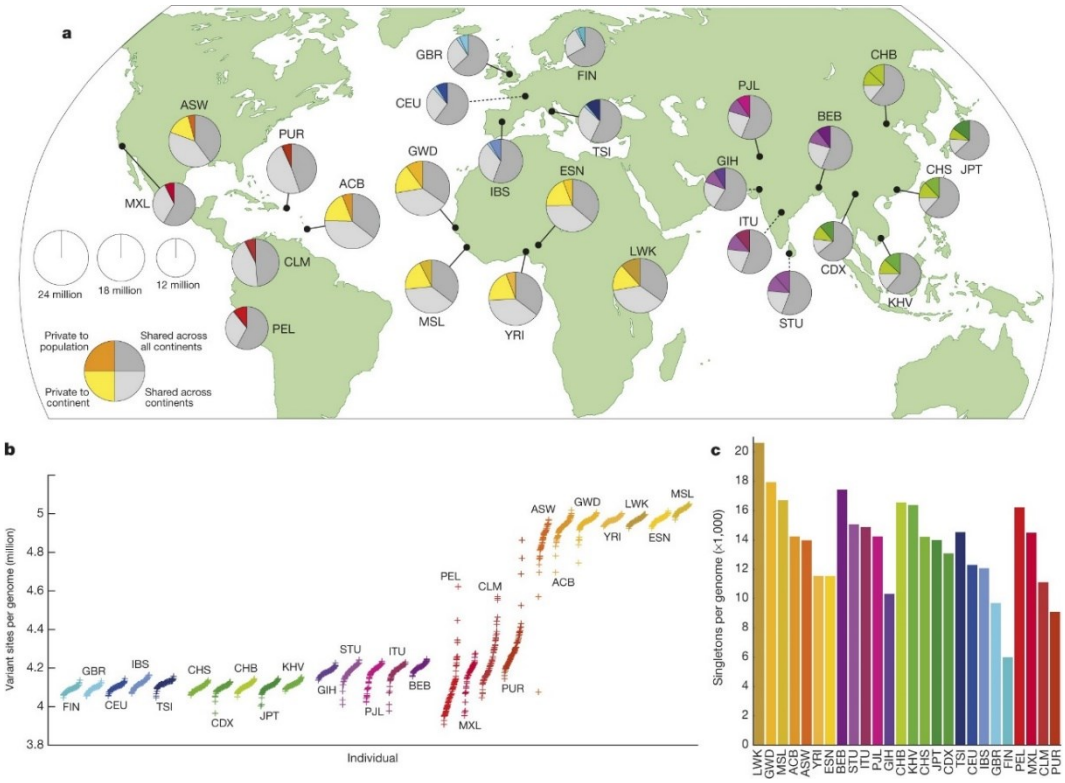


Figure 11. Population Sampling for 1000 Genome Project (1000 Genomes Project Consortium & others, 2015).

3 Methods for off-targets enumeration and targets analysis

This section presents and explains the two software object of the thesis. How the software works, computationally speaking and with a focus on the result processing.

Section 3.1 describes the algorithms behind CRISPRitz (Cancellieri et al., 2020) and therefore CRISPRme, since CRISPRitz is the core processor for CRISPRme indexing and search tasks.

The following sections will include the analysis done in the pre-development phase, necessary to choose the best algorithms and the best data-structure to use in the implementation. This section also describes the practical implementation and the solutions used in the development of the software.

Section 3.2 presents CRISPRitz from a user point-of-view, explaining the tools contained in the set, the mandatory and optional input and how the tools can be used to perform a search from scratch.

Section 3.3 describes the analysis of alternative targets done with CRISPRme. Including all the phases of the post-processing analysis, starting from raw CRISPRitz targets to the final file, containing information about the variants, like the allele frequency and the set of individuals sharing the specific variant(s). This section also describes details about the implementation and operational choices made during development.

3.1 String and Pattern matching algorithms tested and implemented in CRISPRitz

The CRISPR/Cas study is very interesting for doing genetic research. However, we want to treat it also as a computer science problem. As mentioned previously, we may consider the CRISPR/Cas work as a string searching problem, in which the aim is to research one or more pattern inside of a text. The main components used by a string-matching algorithm are the following:

- Alphabet (∑) is an unrepeated finite set of symbols, called also characters, which may appear in every string of the language. When we are discussing biology, we may consider the alphabet as ∑ = {A; C; G; T} for DNA or ∑= {A; C; G; U} for RNA.

- String over an alphabet is a finite sequence of characters from ∑. For the CRISPR/Cas problem we want to use a specific type of string searching called pattern searching. The main characteristic of this type of problem is that the string may be split into two different classes according to their use:

    1. Pattern (guide RNA) is a short sequence of characters, and we are interested to detect whether it appear in a text and/or count how many times it emerges. In biology, the pattern is a short sequence of DNA. Note that the guide RNA is written as a sequence of DNA to avoid the conversion phase from RNA to DNA in the algorithms.

    2. Text (genome) is a large sequence of characters, in which we desire to search the pattern. In biology, the text is the DNA sequence of the genome of interest.

We may face the pattern searching problem in two different ways: exact pattern matching, which involves that the pattern is searched exactly in the text, or inexact pattern matching, which is the case when the pattern is searched allowing some mistakes during the pairing.

The algorithms used for the pattern searching changed based on the context of use and the analyzed data.

For exact string matching there are algorithms optimized to search a single pattern in a text and others optimized to search a set of patterns.

An example of string-matching algorithm to search one pattern is Naive string search algorithm. It screens the whole text searching the pattern using a brute force approach.

Naive algorithm has a very simple logic, but it is underperforming because it uses a basic logic: it slides the pattern

over text one character by one and compares each time the pattern with the current substring of the text.

Whenever a mismatch is found between pattern and text during the comparison or the pattern matches completely with the current substring of the text, the algorithm then slides the pattern by one character (see Figure 12).

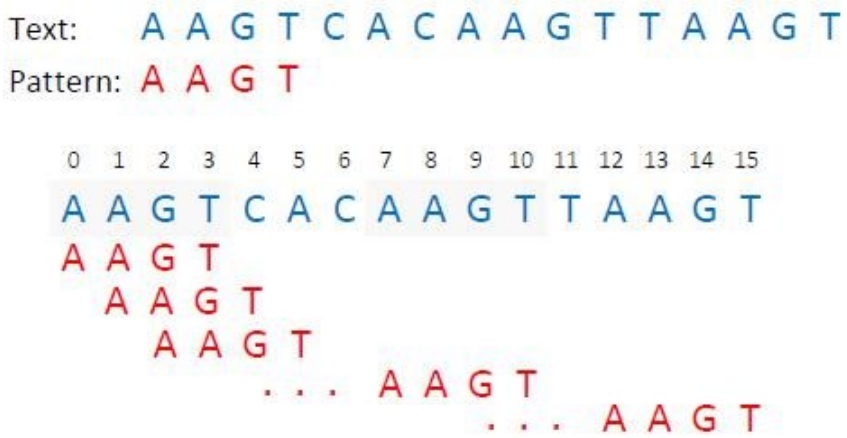


Figure 12. Example of Naive algorithm process.

Naive algorithm is useful to understand the basic idea of string matching. However, it is not usually used to solve real problems. Conversely, Boyer-Moore string search algorithm (Boyer & Moore, 1977) is a very efficient algorithm and it has also been used as a benchmark for the exact string search in literature (Hume & Sunday, 1991). Boyer-Moore algorithm exploits a preprocessing on the pattern to be able to skip characters during the matching between pattern and text. Therefore, it works following two steps (see Figure 13):

- Preprocessing: during this step, the algorithm analyses the input pattern calculating two tables:

    1. $\underline{delta_1}$ that has an entry for each character in the alphabet. Here, the value of each entry corresponds to the maximum index of that character in the pattern, otherwise the length of the pattern.

    2. $\underline{delta_2}$ that has an entry for each character in the pattern. Here, the value of each entry corresponds to the rightmost plausible recurrence of a terminal substring of the pattern.

- Matching: during this step, the algorithm searches the pattern in the text starting from the last character of the pattern and if the characters of the text and the pattern are the same, then it checks the previous character. Otherwise, the algorithm skips a number of characters equal to the maximum value between delta1 of the character of the string and delta2 of the character of the pattern.

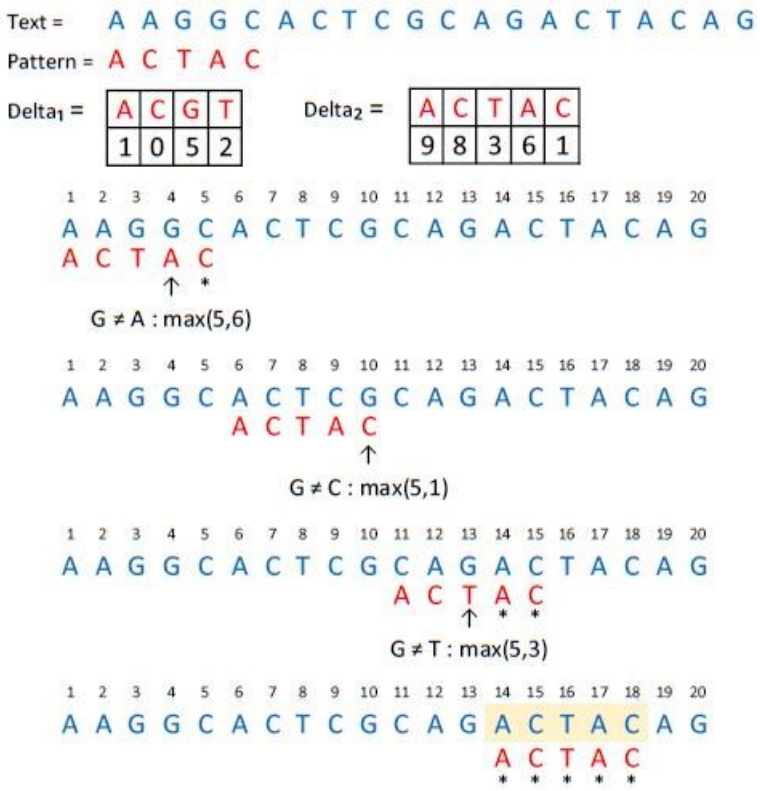


Figure 13. Example of Boyer-Moore algorithm works.

A different approach for pattern searching is to use algorithms that are optimized for more than one pattern. The algorithms that use this approach search more patterns at the same time. An example of this algorithm type is Aho-Corasick algorithm (Aho & Corasick, 1975) that builds a graph starting from a set of input patterns, so it is able to locate all patterns by screening the text only one time. Aho-Corasick is the algorithm that we chose to use for the proposed solution, and we will explain more in depth in the next section.

Until now, we have described only solutions about exact string matching. However, for the CRISPR/Cas problem we are interested to analyze algorithms that allow to have also an inexact matching between pattern and text. In this case we want a more flexible algorithm able to recognize strings with a maximum specified number of errors because the single exact match (on-target) given by a test of a guide RNA of a CRISPR/Cas system is not enough informative. We need to insert the exact match solution in a context with other inexact match solutions (off-targets), so as to have a complete information about the behavior of a CRISPR/Cas system in a specific genome.

We use the Aho-Corasick algorithm to perform the search for the PAM, the sequence we need to start our search.

In the next sections we will explain the two algorithms chosen to implement the proposed solution by this thesis, which are Aho-Corasick and Brute Force Search.

Therefore, in section 3.1.4 we will illustrate how we modified these two algorithms to adapt them to our problem.

3.1.1 Aho-Corasick algorithm

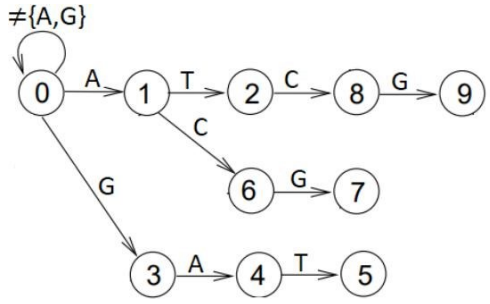
Aho-Corasick is an efficient algorithm used to locate all occurrences of any finite number of patterns in an arbitrary text. The main structure built by the Aho-Corasick algorithm is an efficient finite state pattern matching machine, which is constructed quickly and simply from a restricted class of regular expressions, namely those consisting of a finite set of patterns. Aho-Corasick combines the ideas in the Knuth-Morris-Pratt algorithm (Knuth et al., 1977) with a finite state pattern matching machine. The machine is represented as a graph, which may be drawn as a rooted directed tree.

The pattern matching machine for a set of patterns is a graph which takes as input the text and produces as output the locations in the text at which patterns of the set of patterns appear as substrings.

The Aho-Corasick algorithm consist of two parts. In the first part it constructs the finite state pattern matching machine starting from a set of patterns. Therefore, in the second part it applies a text as input to the pattern matching machine. Finally, the machine signals whenever it has found a match for a pattern.

Before explaining how Aho-Corasick works, we will describe the finite state pattern matching machine. The machine is composed by a set of states where each state of the graph is represented by a number. The machine processes the text by successively reading the symbols of the text, making state transitions and occasionally emitting output. The behavior of the pattern matching machine is dictated by three functions:

- *Goto function g* is a directed graph (see Figure 14) that maps a pair consisting of a state and input symbol, into a state (*current state=0, machine read A: g(A,0) =1*) or the message fail (*current state=1, machine read A: g(A,1) =fail*), where 0 is the start state. In the start state, there is a *goto* function that loops on the root whenever it reads a letter that does not lead the change of state.
- *Output function out* is an array (see Figure 14) that maps a state in a set of patterns that are located from the graph whenever it arrives in that state (*current state=1, machine read T: g(T,1) =2, out (2) =AT*).
- *Failure function f* is an array (see Figure 14) used to complete the directed graph that maps a state into another state and it is consulted by the graph whenever the *goto* function reports fails (*current state=1, machine read A: g(A,1) =fail, gf(1)=0; new state=0*).



(a) Goto function

i	output(i)
2	{AT}
5	{GAT, AT}
7	{ACG}
9	{ATCG}

(b) Output function

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3

(c) Failure function

Figure 14. Example of Aho-Corasick algorithm with set of patterns: {AT, GAT, ACG, ATCG}

Aho-Corasick exploits the pattern machine to analyze the input text. To make the text analysis, Aho-Corasick machine is constructed in two parts:

- In the first part it determines the states and the *goto* functions. *Goto* function is defined by the *goto* graph that is constructed by adding all patterns of the set to the graph. The algorithm begins with a graph of one vertex that represents the state 0. It then includes each pattern into the graph by adding a directed path, which begins from the start state and each state of the path corresponds to a character of the pattern. New vertices and edges are added to the graph unless a new pattern follows a path already present in the graph. The new pattern is added to the output function of the state at which the path terminates. Finally, Aho-Corasick adds a *goto* function that is a loop from state 0 to state 0 of all input symbols other than symbols that lead the change of state 0 (see Figure 14).
- In the second part the algorithm computes the failure function (see Figure 14). It is constructed starting from the root and is computed for each vertex. *Failure* function is calculated after the *goto* function because the algorithm needs to the *goto* function to build *failure* function. To clarify this part we define *depth* of a state *s* in the *goto* graph as the length of the shortest path from the root to the state *s* (*current state=8: depth(8)=3*) and the state *r* is the state of depth *d-1*. The failure function is built analyzing together all states with the same depth. It starts from all states of depth 1, which have *failure* equal to zero (*current state=1: f(1)=0; current state=3: f(3)=0*). Therefore, it continues to build the failure function following these steps:

  1. Starting from the current state *s*, the algorithm calculates the result of the *failure* of the previous state *r* and sets the result as the new current state (*current state=4: f(3)=0; new current state=0*).

  2. Reset the current state with the result of the *failure* function, until the *goto* function returns a state different from fail (*state=0: g(0,A)=6 fail*).

3. The algorithm calculates the result of the *goto* function of the current state and sets it as the result of the *failure* of the initial state *s* (*current state=0, initial state=4: f(4)=g(0,A)=1*).

During the computation of the failure function, the algorithm also updates the output function. Whenever it determinates *f(s) = s'*, where s' is the last state of a stored pattern, it adds to the output of the state's the output of the state s' (*s=5, s' =2: f(5)=2: out(5)={GAT, AT}, out(2)={AT}*).

Now the machine pattern matching is ready to analyze the text (see Figure 14). The input text is screening following an operating cycle, where s is the current state of the machine and the current symbol of the input text *x* (Figure 15):

1. If *g(s,a)=s'*, the machine makes a *goto* transition. It enters into the state *s'* and the next symbol of *x* becomes the current input symbol.

In addition, if output *(s')≠empty*, then the machine emits the out (*s'*) along with the position of the current input symbol.

2. If *g(s,a)*=fail, the machine consults the *failure* function f and is said to make a *failure* transition. If *f(s)=s'*, the machine repeats the cycle with *s'* as the current state and a as the current input symbol.

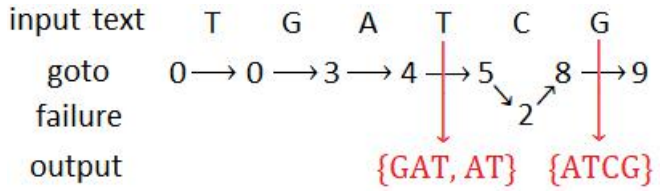


Figure 15. Example of Aho-Corasick using machine of see Figure 14.

3.1.2 Brute Force Search Algorithm

Brute Force Search is a very simple algorithm used to search a substring in a string, as explained before (Naïve Search Algorithm), Brute Force Algorithm (see Section 3.1.2), just search for the proposed string (guide RNA) on the genome we have before indexed with Aho-Corasick for PAM. Practically, the algorithm takes the array list of PAM created in the Aho-Corasick step, then uses this information to search on the genome.

So, it takes the proposed RNA guide and search on the genome, starting from the first PAM position in the array list, after that, continue to search using all the subsequent indexes.

After that, the algorithm takes the second RNA guide and repeat the process.

This process is executed specifically in this way:

1. Algorithm takes the first guide of the input file, inserted by the user.
2. Algorithm start searching the genome, using as index, the first index found in the PAM array list, an array containing the list of all the PAM indices found in the current genome.

3. The algorithm keeps doing this search, comparing every character of the RNA guide with every character of the genome (every character is before converted in a bit code).
4. After processing all the PAM indexes with the first guide, the algorithm takes the second guide from the input list and repeat the process.
5. The process continues until the end of the guide input list.

In this step, the software, also collect information about the guides, like, total off-target per guide and mismatch per BP.

This process is very simple, but the efficiency of the search cannot be improved other than changing the used hardware.

In the next chapter we will present the total implementation, staring from the reading of the input files and describing all the operations performed by the software until the writing of results files.

3.1.3 Ternary Search Tree

Ternary search tree (Bentley & Sedgewick, 1998b) is a very simple and fast data structure, which is able to solve many string search problems and it supports a large range of useful operations.

The real power of ternary search tree is the combination between the time efficiency of digital tries for the proceeding of each character, with the space efficiency of binary search trees (Bayer & McCreight, 1972), with the only difference that ternary search tree has three children instead of two.

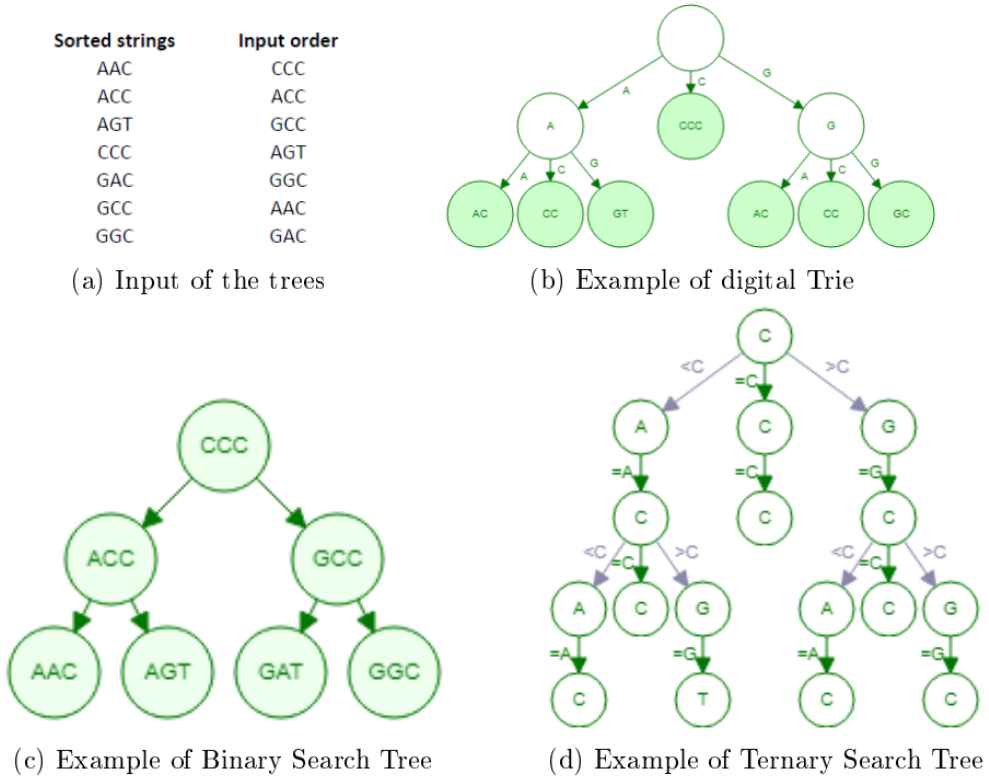


Figure 16. Examples of ternary search trees.

The basic idea of the ternary search tree research is that it compares the current character of the input string with the character of the node.

If the current character is lexicographical smaller than the character of the node, the research goes to the left child. On the contrary, if the current character is lexicographical greater than it, the research goes to the right child. Finally, if the current character is equal to the character of the node, the research goes to the middle child and ternary search tree proceeds to compare the next character of the input string.

The ternary search tree algorithm searching is done following two steps:

- During the first step, the algorithm reads the input text (or list of strings) and builds the ternary search tree by inserting string by string the whole input text.
- During the second step, the algorithm searches the input pattern (or list of patterns) in the built ternary search tree. The flexibility of this data structure

is that the user may perform exact research or setting up research allowing to have one or more mismatches between the input string and the found string.

The construction of the ternary search tree is performed by the insert function that inserts an input string (word), which is taken from the input text, into the tree if it is not present in the tree, otherwise, the function does nothing.

When the insert function inserts a new string in the ternary search tree, it starts from the root following the path that has the longest common prefix with the input string.

The function follows the path until the whole new string is inserted in the ternary search tree or the longest common prefix between path and new string is finished.

In the last case, the function checks if there is a middle child of the current node, if it is not present the function creates new consecutive middle children until the end of the input string.

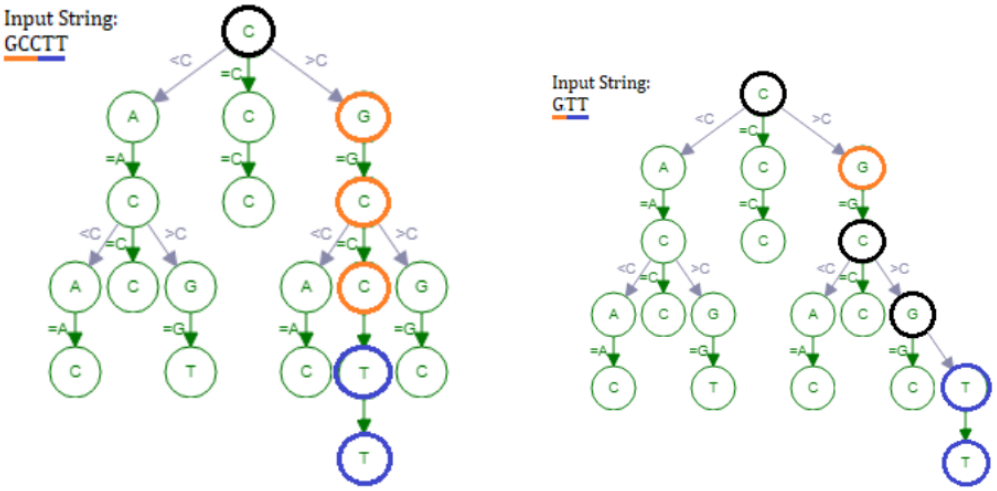
On the contrary, if the last character of the path of the common prefix has a middle child, the function moves into the middle child and compares the current character of the input string with the character of the new current node: if the current character is lexicographically smaller than the character of the node a left child will be created, otherwise, a right child will be created.

The order of the string insertion influences the structure of the tree and the insertion speed of the strings. This ternary search tree characteristic is not so sensible like in binary search tree; however, it would be important to have a balanced final tree.

Therefore, before inserting a list of strings in the ternary search tree, it would be appropriate to sort the string list. Afterwards, the best way to insert a sorted list of strings is to insert the string in the middle at first, going to the left and right sub lists and inserting the strings in the middle of these sub lists, continuing in this way by inserting each time the string in the middle of the sub list.

The ternary search tree provides several different searches. The main research that we are interested to analyses are two Membership Searching and Near-Neighbor Searching. Membership searching is a recursive function that makes an exact search of a pattern into the ternary search tree. The function starts from the root and compares character by character the input string with the nodes of the tree. Membership searching compare a character of the pattern moving into the tree until it finds the same character in a node.

The function then moves into the next character of the pattern and goes to the middle child of the node. The function stops when it finds the pattern or when it arrives to a leaf.



(a) Creating new the middle children. (b) Creating a new left child.

Figure 17. Examples of insert on ternary search tree of figure.

Near-Neighbor Searching is a recursive inexact search function. The basic version of the function allows to decide a Hamming distance that indicates the maximum number of mismatches between the input pattern with the found string. The logic of Near-Neighbor Searching is very similar to Membership searching. The only difference is that when the first function compares two characters, if they are different the function checks if the number of mismatches allowed is greater than zero. In case of more than zero remaining mismatches, the function decreases the mismatch counter and continues to compare the next character of the pattern with the character of the middle child node. In the other cases Near-Neighbor Searching is the same as Membership searching.

3.1.4 Implementations details for presented algorithms

In this section we want to describe CRISPRitz (Cancellieri et al., 2020), which is the proposed solution to predict the off-target activity of a guide RNA applied on a genome and perform an analysis of the guides profiles.

Our algorithm is based mainly on Aho-Corasick (see Section 3.1.1), Brute Force Search (see Section 3.1.2) and Ternary Search Tree (see Section 3.1.3), which are used to perform the pattern search action with different settings. We thought to implement the CRISPRitz following the logic of Cas-OFFinder: first, searching the input PAM on the genome, create an array list containing all the PAM indices founded by the Aho-Corasick Algorithm step and using this list to perform an ex-novo search on the target genome, using Brute Force Search Algorithm.

The following steps will explain how each phase is implemented.

The first step accomplished by the software, is the reading of the user input. So, the software takes in input:

- List of guides the user want to use to search the genome
- Reference genome, divided in FASTA files (one for chromosome)
- PAM (sequence used to search target on the genome)
- Number of threads (for multicore installation)
- Number of mismatch (threshold for the search)
- Result writing (a character activating the results writing)

After this collection of input, the software start reading all the needed information.

First of all, we implemented the main function, in which we start all the needed functions.

The first function is the reading pam, a simple C++ function, we use to read the PAM from a txt file, we read the PAM string, we uppercase all the PAM characters, just in case the user forgot it, and we save the string in a variable we need for the following searching on the genome.

After that, we proceed reading the guides from the input guide files.

This operation is similar to the reading pam operation, but in this case, we perform some processing on the guides.

In fact, we convert all the characters in the guide string, in a bit code composed by 4 bits.

An example,

GCTCAG → 0100|0010|1000|0010|0001|0100

1 This operation, permit us, to codify all the guides and the genome in a bit code, very useful to perform faster searches and to let us use wild characters in the genome or in the guides. Like, IUPAC wild-characters, codified using the bitwise sum of their 4-bit code representation. For example, (D → A, G, T), so the summed 4-bit code will be A (0001) + G (0100) + T (1000) → 1101. This is a very fast and compact way to represent the possible ‘undefined’ characters on the genome or the input RNA guides.

After reading guides, we start generating prime numbers, those prime number are necessary to identify which guide is writing on the results file, to avoid a possible overwriting if two guides find a target in the same PAM index.

After all those readings, we dynamically generate the profile matrices, vector of string and integers, that we need to collect all the information about mismatches, off-targets and so on.

At this point, we start the reading and converting the genome.

This step is summarized in the following points:

- Read the genome (one chromosome at time)
- Convert the chromosome from string to 4-bit

After those reading, we proceed to scan the genome, to find all the PAM indices, as we will describe in the following paragraph.

The second main step is the PAM searching and index creation.

After all those reading, we start processing the actual genome we saved in the RAM variable.

So, we start use the previous explained Aho-Corasick Algorithm, to perform a rapid and efficient search of all the PAM indices on the genome.

We start using the PAM we read from the file, for example:

'NNNNNNNNNNNNNNNNNNNNGG 3'

This is a possible PAM sequence we can read from the input file, the number '3', it's a count that our software needs to understand how long the pam is, in fact, the PAM we need is:

'NGG'

So, we pass the PAM sequence to Aho-Corasick Algorithm.

The implementation of Aho-Corasick was taken as a just-made implementation found on the internet, it's developed in C++, and we just modify little parts to our use.

All the changes are listed in the following list:

- Admitting uppercase character
- Dividing all the search in two separated results arrays
- Create all the possible combination of characters starting from wildcards

The C++ implementation use a rapid way to identify a correct character, use a bit a bit subtraction and calculate the result, to check if the character appertains or not to the input string, in our case all the bit a bit operation had to be done with uppercase characters, so change the bit a bit operation to perform a correct calculation with uppercase characters.

Aho-Corasick, is created to find all the occurrences of a desired pattern in a given text, but we need a slightly different operation, we need to divide the results based on where they were founded on the genome, in fact, when we search on the genome, we also perform a reverse search, so we search for the reverse complement of our PAM.
For example,

'NGG' would also need to be searched as 'CCN'

So, we need to collect the indices of the PAM based on which PAM is used to find the result, and we created a very simple code to divide results from the 'NGG' search and the 'CCN' search.

```
for (j = 0; j < k; j++)
{
    if (out[currentState] & (1 << j))
    {
        if (j < mez_K)
        {
            pamindices[i]=((i-(pamlen-1))>0) ? (i-(pamlen-1)) : 0;
            if((i-(pamlen-1))==0)
            {
                pamindices[i]=-1;
            }
            break;
        }
        pamindicesreverse[i]=((i-(pamlimit-1))>0) ? (i-(pamlimit-1)) : 0;
        if((i-(pamlimit-1))==0)
        {
            pamindicesreverse[i]=-1;
        }
        if(pamindicesreverse[i]>(genome.size()-guidelen))
        {
            pamindicesreverse[i]=0;
        }
        break;
    }
}
```

Figure 18. PAM search on positive and negative strand. Accomplished by reversing complementing the PAM sequence.

As we can see from the code, we divide our results based on which PAM found the target on the genome.

To explain this, I recall the third point of our list:

Create all the possible combination of characters starting from wildcards.

To create all the possible combination, we create a short function, that takes the PAM inserted by the user, for example:

'NGG'

And compute all the possible combination, so:

AGG, CGG, GGG, TGG

And all the reverse complements:

CCT, CCG, CCC, CCA

Then, we save all those PAM in vector of string and we use them as explained before (see Figure 19).

After this point we proceed with the search, using the previous explained, Brute Force Search Algorithm. In the next paragraph we will explain the genome search.

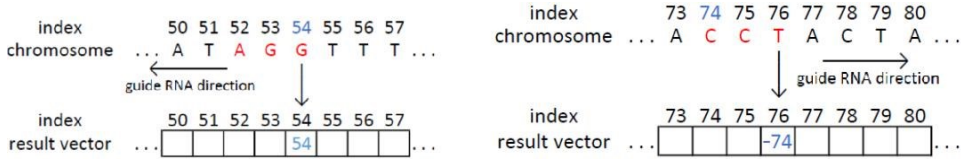


Figure 19. Example of parallel Aho-Corasick procedure with NGG sequence.

In this paragraph will be explained what the genome search is and how it is implemented.

We implemented this part of the software taking inspiration from Cas-OFFinder but adding some modifications that we need to add the features we explained before.

In fact, our search start with the creation of our memory vector, needed to save results and to perform a posterior analysis to obtain profiles and counts.

The summarized steps:

- Creation of results array lists (positive and negative, for saving all the possible results)
- Guide targeted lists (containing all the guides prime numbers to avoid loss of results)
- Split of guides array (avoiding memory out of bound problem)
- Two search engines (one for positive PAMs and one for negative PAMs)

The first step was simple to realize and it's easy to explain. We create two vectors in which we can store all the needed information to save results and to compute profiles.

The second step consists in create a vector, that can store increasing prime numbers that we need to avoid loss of results, this problem must be addressed, because during tests we find that some guides found off-target in the same PAM index, so we need to store those guides in this way, to avoid a lot of memory consume and to maintain the storing compact.

The third step consist in the creation of a splitting system, that divide the total number of input guides in part containing 100 guides, this operation, avoid memory out of bound and avoid big writing on the results file, so it's a faster and better performing solution.

In the fourth step, we proceed with the real analysis, in fact we perform two searches, one using the positive PAMs collected in the previous phase and the other search using the negative PAMs. So, we can divide results and we can now from which strand of the genome, comes the off targets.

In this step, we search the genome using the Brute Force Algorithm (see Section 3.1.2), starting with the first index in the PAM list, we take the first guide in the guide list, we use it as a pattern and we check, character per character, if the guide match with the genome substring.

We use a mismatch threshold to set a cap we don't want to exceed during this process.

The process is repeated using all guides in the list, after we complete this, we take the second index in the PAM list and we restart the process, and so on until end of all the PAM lists.

The following paragraph will explain how the implantation was parallelized.

Our competitor, Cas-OFFinder, uses an open platform and SDK to perform its parallelization, OpenCL (Munshi, 2009a).

This platform is well known for its portability, in fact, the main goal of OpenCL, is to permit the execution of the same identical code, on all supported platforms, without the need of any changes in the code.

This approach is very good if you want to open the access to your software to a lot of final users, but has also some cons, in fact, OpenCL need some pre-installation, not very simple for a naïve operator, such a biologist without any experience in Unix platform or usage of terminal commands.

So, we try to overcome this inconvenience using a very distributed platform, that can be used on every Unix platform with a GCC compiler.

In fact, our software reaches its parallelization with the use of OpenMP, a very reliable and well-known API (Dagum & Menon, 1998).

The package API is pre-installed in every GCC compiler of Unix distribution, starting from the version 4.7, one of the most spread versions of the popular C++ compiler.

To obtain the parallelization with OpenMP API, we need first to understand correctly how it works and what we need to change our code to adapt the parallelization.

We need to change the approach we have to the analysis, in fact, the parallelization, need safe threads accesses to maintain the output of functions in line with the serial execution of the code.

In fact, in the first attempts to parallelize the code, we encounter some problems with writing on files and with the overwriting of results during execution.

We need to guarantee a safe access to writing on RAM variables and disks, so we start trying using blocks and queue.

But this approach was very ruinous for the performance of the code, we obtain a very little improving in terms of speed and so we need to develop another approach.

So, we start thinking how we can keep all the information we need, without breaking the safe thread need and without interrupting cycles, to obtain the maximum speed-up we can gain.

After some testes we end up with this code:

```
#pragma omp parallel for schedule(static) private(j,guidecount,currentmissmatch) num_threads(threads)
for (i=0;i<pamnegsize;i++)
{
    for(guidecount=inizio;guidecount<fine;guidecount++)
    {
        currentmissmatch=0;

        for(j=pamlimit;j<guidelen;j++)
        {
            if((genomebit[j+pamindicesreverse[i]]&reverseguidesbit[guidecount][j]) == 0000)
            {
                currentmissmatch++;
            }
            if(currentmissmatch>missmatchthrestotal[guidecount])
            {
                break;
            }
        }
        if(currentmissmatch<=missmatchthrestotal[guidecount])
        {
            resneg[i]=pamindicesreverse[i];
            guidefoundneg[i]*=primenumbers[(guidecount%100)];
        }
    }
}
```

Figure 20. Parallel code for guide search on the genome.

As you can see, with this implementation we never break the outer cycle, the parallel one and we never write on files, we only use variable allocated before execution to avoid exiting from a thread safe condition and to avoid overwriting in memory.

Now, explanation of the for-loop cycle parallelized with openMP constructs.

```
#pragma omp parallel for schedule(static) private(j,guidecount,currentmissmatch) num_threads(threads)
```

Figure 21. Line of code of openMP construct to use 'parallel for' API.

This for cycle is adapted to use the OpenMP API instructions:

- #pragma omp, instruction needed to inform the compiler we want to run this cycle in parallel
- schedule(static), instruction we had to the for cycle to inform the compiler we want a schedule based on a static assignment, so no changes during execution
- private(#variables), instruction added to inform the compiler that every thread in the for cycle, need to have a copy of those variables, in this way we prevent racing condition and overwriting
- num_threads(integer number),instruction that informs the compiler that we want to use #number thread, so our cycle we will ask the master thread to reserve those threads for execution

With this implementation, we solve a lot of problems in term of stability and performance, in fact, the code runs using all the threads we reserved at maximum capacity, without needing to stop for writing or for resetting some variables or assigning values to shared variables, all we need to save is stored in arrays created to maintain all the possible solutions, without need to stop the execution or causing waiting between threads.

In this paragraph we want to describe the Ternary Search Tree implementation, which is the proposed solution to predict the off-target activity of a guide RNA applied on a genome. Our algorithm is based mainly on Aho-Corasick and Ternary search

tree, which are used to perform the pattern search action. We thought to implement the CRISPRitz following the logic of Cas-OFFinder: firstly, searching the input PAM on the genome and save the target sequences and then comparing the input guide RNA with the saved sequences.

Our initial idea about the implementation was to use only Aho-Corasick.

Being Aho-Corasick an algorithm that allows to search a set of string, we thought to generate a set composed of all possible strings starting from the input guide RNA, following the mismatches constraints, and build a pattern

matching machine based on the generated set. For instance, if we want to generate a machine on the guide RNA: GAGTCCGAGCAGAAGAAGAA,

allowing one mismatch, the set will be composed by:

- GAGTCCGAGCAGAAGAAGAA
- CAGTCCGAGCAGAAGAAGAA
- AAGTCCGAGCAGAAGAAGAA
- TAGTCCGAGCAGAAGAAGAA
- GCGTCCGAGCAGAAGAAGAA
- . . .
- GAGTCCGAGCAGAAGAAGTC
- GAGTCCGAGCAGAAGAAGAA
- GAGTCCGAGCAGAAGAAGAG
- GAGTCCGAGCAGAAGAAGAT
- the reverse complement of all previous guide RNA.

It is possible to see that just allowing one mismatch on a guide RNA 20bp long, the set generation is not trivial because the number of patterns to build the pattern matching machine is very high (122 in the proposed example):

$$\text{totalGuides} = \left[\left(\sum_{i=1}^{\#mm} \binom{gRNA}{i} * 3^i\right) + 1\right] * 2$$

in the previous formula we sum the number of generated guides RNAs allowing *i* mismatches, where *i* starts from 1 to #mm that is the maximum number of mismatches allowed.

Each time we calculate all combinations of the positions where we will have the mismatches with the binomial. Then, we multiply it with the number of combinations of the possible mismatches 3^i, where 3 is the number of nucleotides considered for the mismatches and *i* is the number of mismatches allowed.

We add 1 because we also want to insert the original input guide RNA.

Finally, we multiply by 2 the whole operation, because we need to generate also the reverse complement of each generated guide RNA, because we have to search along both strands of the DNA at the same time.

We performed an initial test in python (see Table 1) for the generation of the string set, the building of the pattern machine and the test on the chromosome 2 of the Human Genome (hg19). From the measured times we deduced that the generation was very influenced from the number of mismatches and the waiting time was already too high when we wanted to generate all guide RNA for 4 mismatches.

It is important to note that, here, we did not consider the DNA or RNA bulges case because they were initial tests.

Building and searching time were acceptable, we then concluded that Aho-Corasick was a very powerful algorithm, but the real problem was the generation of all possible guide RNAs. Finally, we decided to use it only during the PAM searching because the number of all possible generated PAMs it is negligible. PAM do not have mismatches, because the PAM generation is conditioned only by the special symbols (Johnson, 2010) that it may have.

For instance, if we want to search the SpCas9 PAM (NGG) on a genome, we will generate just 8 sequences: AGG, CGG, GGG, TGG, CCA, CCC, CCG and CCT.

MM allowed	Generation (s)	Machine building (s)	Searching on chr2 (s)
2	0.000001	0,000001	0.219
3	0.125	0.000001	0.297
4	2.047	0.047	0.3
5	23.622	0.437	0.449
6	211.263	4.534	0.551

Table 1. Times about guide RNA (GAGTCCGAGCAGAAGAAGAA) by using Aho-Corasick (python version).

Initially, we wanted to implement CRISPRitz using python language, because the main idea was to provide a versatile tool that all users were able to modify according to their needs. However, after the first results we decided to implement our software in C++ because the computational time was too high if compared with the computational time of Cas-OFFinder, which is written in C++ and OpenCL.

The implementation of CRISPRitz is divided in two parts: in the first part, the software takes as input the PAM, searches it into the input genome, builds the ternary search tree with the saved sequences and then CRISPRitz save through serialization the ternary search tree into a binary file. In the second part CRISPRitz reads the ternary search tree from the binary le and the software then searches into it the input guide RNA following the user-defined constraints (number of RNA/DNA bulges and number of mismatches allowed). Whenever we execute the first part of CRISPRitz, the software generates and saves a ternary search tree for a PAM and a genome. If we want to use the o-target activity prediction function, we may execute the second part of CRISPRitz every time we want, without execute the first part every time.

We chose to split the implementation in two parts for two reasons: software preprocessing time was very high, which starts from the reading of the genome until the ternary search tree is ready for the guide RNA matching.

The second reason is that the algorithm uses too much RAM during the preprocessing, and it is not guaranteed that all personal computers may provide enough memory for this step.

Computational time and used space are two fundamental characteristics that we want to optimize with our implementation, because we have designed CRISPRitz for a personal computer of a biologist and we supposed that the common user has not performing computer available.

Therefore, our idea was to process the input genome and the input PAM to generate the ternary search tree, so the final users may use just our "preprocessed genome" to make their analysis.

Finally, during all the implementations, we clashed with a very hard tradeoff between improving the space performance or improving the time performance.

In the following paragraph, we will describe the part of CRISPRitz that produces the "preprocessed genome".

CRISPRitz works by reading one chromosome at a time and making its operations on the read chromosome. The implementation starts with taking as input the desired PAM and the chromosome of the genome.

Before any operation, it is fundamental that the software applies the "toUpperCase" method on the input data. It is not a problem for the input PAM, because PAM is not so long that this operation does not influence the total time.

However, applying "toUpperCase" on a chromosome may be a problem.

The solution is to import the OpenMP library that allows to parallelize our implementation by using the parallelized algorithm provided by openMP. Afterwards,

CRISPRitz generates all possible PAMs starting from the input PAM.
The PAM set is generated by the generatePAM function in two main steps:

1. *SwitchSimbol*, is the function in which the special symbols of the input PAM are converted into nucleotides. Before that, the function splits the PAM into an array of characters, therefore, it analyses each character of the array and converts each special symbol into a string of the corresponding nucleotides. For instance, in case of NRG as input PAM, *switchSymbol* split the PAM into an array: [‘N’, ‘R’, ‘G’]; then it converts special symbols into a string of nucleotides: [‘ACGT’, ‘AG’, G’].
2. *getProducts,* is the function that combines the characters of the strings of the array generated by *SwitchSimbol,* to produce all possible PAM. For instance, the function reads the array of string: [‘ACGT’, ‘AG’, G’]; then it picks on character from each string and produces all ordered combinations: ["AGG", "CGG", "GGG", "TGG", "AAG", "CAG", "GAG", "TAG"]

The two previous steps are repeated twice, because we need to generate a set that contains all possible PAM for both strands of the chromosome.

Therefore, *generatePAM* generates the reverse complement of the input PAM (for instance, the reverse complement of "NRG" is "CYN") and applies the two steps on it and merge the two arrays.

The array generated by the *generatePAM* function is used as a set by the *searchPAM* function that implements the Aho-Corasick algorithm. The workflow of the *searchPAM* function is very similar to the workflow explained in the Aho-Corasick section. The preprocessing, where the matching machine is built, is the same to the theoretical version explained previously.

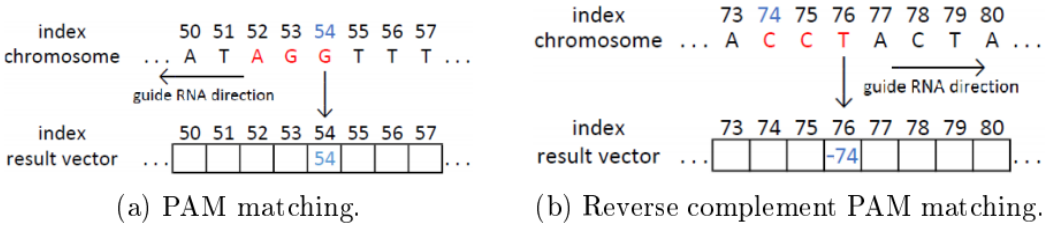
There are some differences during the process of searching done by the matching machine between the theoretical version and our implemented version.

The first difference is that our implementation has a basic parallelization, because we want to have the fastest implementation possible. The function then splits the for loop on the chromosome into a variable number of for loops depending on the number of available threads. Each for loop analyses a substring of the chromosome, so, the machine may search simultaneously in more parts of the chromosome.

Basically, each thread has its own variable for the current state of the pattern matching machine, an index of the substring of the genome and an index on the result vector.

The second difference is what we save in the result vector, because in the original version Aho-Corasick reported the recognized string whenever that it found one, instead here we need to save all indices where we find the PAM.

Therefore, the vector is as large as the chromosome because the threads may not write simultaneously in the same entry. So, when a thread finds an occurrence of the PAM, it saves the index at the corresponding position in the result vector (see Figure 22). On the contrary, if they found PAM is the reverse complement of the input PAM, the function saves the index where the PAM matching started, and the index is saved as a negative value (see Figure 22). Finally, the filled vector will have several empty entries that are removed by the "erase" method.



(a) PAM matching. (b) Reverse complement PAM matching.

Figure 22. Examples of parallelized Aho-Corasick saving with NGG PAM.

The vector filled by the *searchPAM* function is used by CRISPRitz,

to retrieve the sequences that will be used for ternary search tree building. During the sequence retrieval a for loop reads each entry of the vector: if the read number is positive, it is saved the chromosome substring of length $22 + |PAM|$ (for instance, with NGG PAM it will be saved substrings of length 25), starting from the position indicated by the index saved minus 22.

Conversely, if the read number is negative, it is converted into positive and the saved chromosome substring will have the same length of the previous case, but the starting position will be the saved index.

The retrieved sequences, which are saved in an array, are then sorted in lexicographical order by a parallel algorithm. When the sorting algorithm sorts the sequences, moves in the same way the corresponding chromosome indices, which were saved previously by *searchPAM* function. This step is important because the order of the strings during the ternary search tree building influences the balancing and the building time of the data structure.

The ternary search tree building uses the same logic proposed by the original version explained in the Ternary search tree section. We modified just some elements to adapt the data structure for our use. The first main modification is the structure of the node object. The nodes of the ternary search tree are saved in an array of nodes and each node is composed by the character of the node and the references of the children.

In the original version, these indices were pointer of 8 bytes, then each node weighed 25 bytes (8bytes $*$ 3children + 1byteschar). If we want to use a ternary search tree to store the target sequences, it is very common to use more than 500 million of characters, therefore, if we need one node per each character, we need at least 12.5 Gigabytes of RAM memory.

We improved the memory required by a ternary search tree modifying the structure of the node. In our version each node is already composed by one char and three child indices, but the child indices are no longer pointer to the children, they are now three int (4 bytes) indices that contain the node vector index value of the entry of that child.

Another modification that we applied to the ternary search tree was to bind the leaves of the tree with an array of PAMs. Considering that we insert only strings 22 characters long, we are sure that each new leaf will not have a middle child during a later string insertion. Therefore, we use the value of the index of the middle child as an index of an array of PAMs that contains the PAM and its chromosome index connected to that string.

If CRISPRitz inserts a string already present in the ternary search tree, the new PAM and its chromosome index will not substitute the previous values, but they will be inserted in another entry of the array of PAMs and it will be bound with the entry connected with the leaf of the ternary search tree like a list (see Figure 23).

Previously, we mentioned that the implementation is split into two parts, where in the first one, CRISPRitz reads the chromosome and the PAM, builds the ternary search tree and saves it in a binary file.

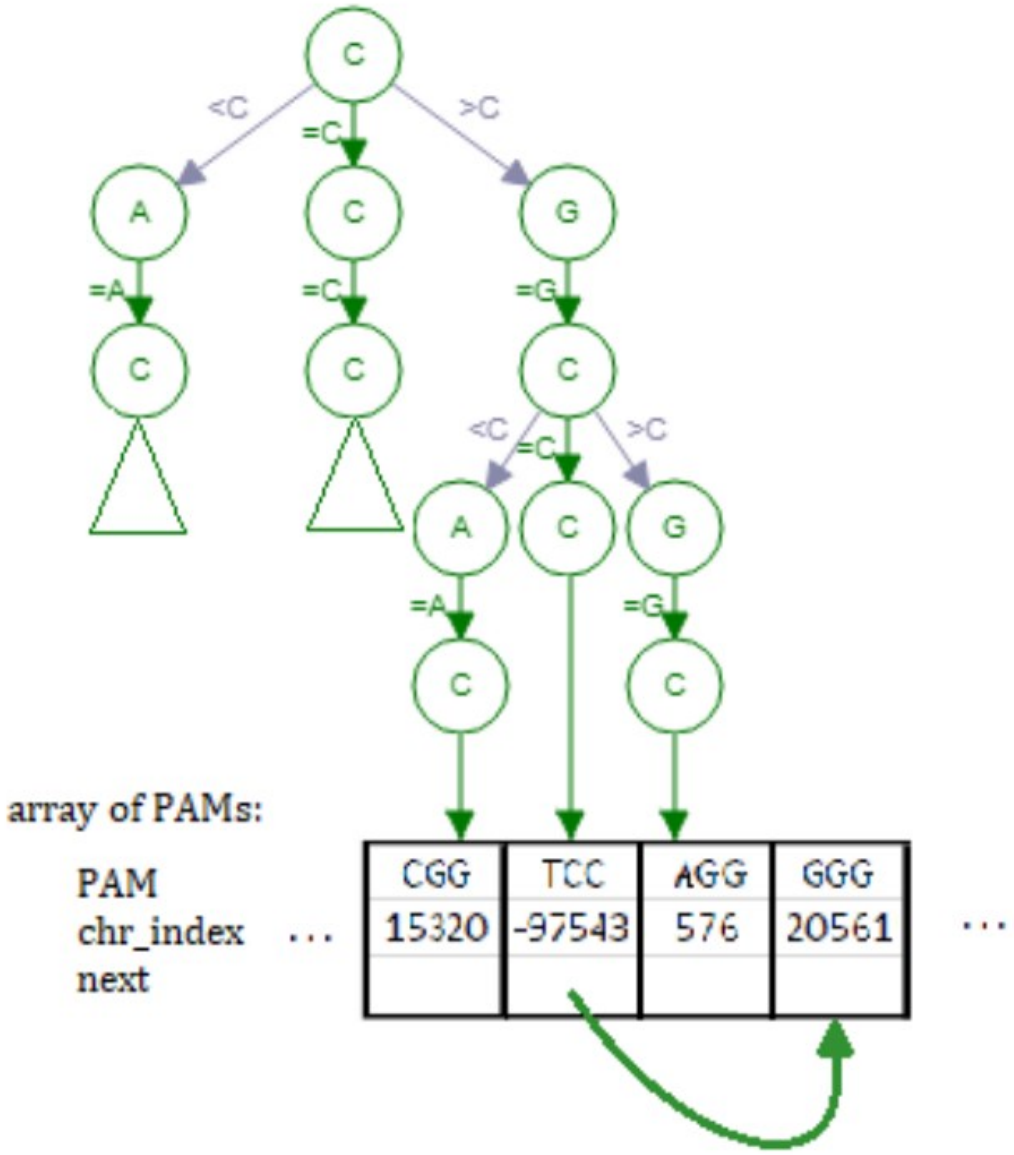


Figure 23. Example of array of PAMs.

In the second part, CRISPRitz reads the binary file and the guide RNA and searches the input guide RNA into the ternary search tree.

After the reading, CRISPRitz provides the most important function useful for the user: *guideMatch*. This recursive function uses the same logic of the tree traversal. *guideMatch* takes as input the root of the ternary search tree and the number of mismatches, DNA bulges and RNA bulges.

The function starts trying to recursively go into the left children and after into the right children.

There are two cases in which *guideMatch* is interested to go into the left of right child: whenever the current character of the input guide RNA is lexicographical smaller (left child) or greater (right child), alternately, when the matching between the input guide RNA and the character path has used less mismatches or DNA/RNA bulges than defined by the user.

Afterwards, *guideMatch* tries to go into the middle child if the current node is not a leaf (leaves may have right and left child, but not the middle child) and it makes a recursion calls in each of the following cases:

1. *guideMatch* compares the current character of the input guide RNA with the character of the current node and if they are the same it saves both in two distinct output strings, where one is for the path (target site) and the other is for the input guide RNA. It calls itself on the middle child with the same parameters, moving into the next character of the input guide RNA. Otherwise, if the two characters are different, the function checks before that if the variable of the maximum number of mismatches is greater than zero and, in this case, it also saves both characters, but the character of the node is saved in lower case. Then, it calls the recursion on the middle child, decreasing the mismatch variable by one.
2. *guideMatch*, checks if the variable of the maximum number of DNA bulges is greater than zero and, in this case, it saves the character of the node and a dash ('-') instead of the current character of the input guide RNA. Therefore, the function calls the recursion on the middle child, decreasing the DNA bulge variable by one, maintaining the current character of the input guide RNA.
3. *guideMatch*, checks if the variable of the maximum number of RNA bulges is greater than zero and the current character of the input guide RNA is not the last character. In this case it saves the current character of the input guide RNA and a dash ('-') instead of the character of the node. Therefore, the function moves into the next character of the input guide RNA and calls the recursion again on the current node decreasing the RNA bulge variable by one. Furthermore, during the second recursion in the current node there is a constraint that does not allow to travel into the left and right child only for that node, because in that precise moment guideMatch is analysing the character of the current node.

Finally, the function saves in a vector the data about the target (strings, number of mismatches and RNA/DNA bulges) whenever the comparison between the last character of the input guide RNA and the character of the current node satisfies the constraints imposed by the first case explained previously.

Note that in the last character may not be a DNA or RNA bulges, because it would be biologically wrong.

The last phase of the second part of CRISPRitz is to write all the obtained results in a text file with the same layout used by Cas-OFFinder. So, we may compare the results given by the two tools in the same way. In our implementation we consider more broadly the definition of DNA and RNA bulges with respect to Cas-OFFinder. When we want to search a guide RNA and we set a threshold for bulges greater than 1, the software considers this value whether as the maximum number of bulges allowed or as the maximum bulge size. Therefore, CRISPRitz returns more putative off-targets with bulges in distant positions.

Another difference between the CRISPRitz and Cas-OFFinder bulge research is that our software may search the RNA or DNA bulges along the whole guide RNA, excluding only the last nucleotide at the 5' end (the nucleotide at the end of the guide RNA that does not bind with the PAM: 5'-ATC...CCNGG-3'). Conversely, Cas-OFFinder has a constraint that does not allow to have a DNA or RNA bulge on the first nucleotide of the guide RNA that binds to the PAM. Cas-OFFinder imposes this constraint because some papers studied the problem of how many and where bulges and mismatches a guide RNA

is able to endure (Lin et al., 2014) and they supposed that there are some spots that does not allow a mismatch or a bulge.

3.2 CRISPRitz: variant-aware off-targets enumeration tool

CRISPRitz was developed to answer a specific request, made variant-aware off-targets enumeration and analysis, accessible, easy and stand-alone.

The software includes all the necessary tools to perform a complete analysis from scratch.

The software is composed of five different tools, each one developed to solve a specific task, like create an alternative genome adding genetic variants to a reference genome or integrating data with functional annotations to better understand if a specific target can be predicted as harmful or dangerous.

In Figure 24 are summarized CRISPRitz functionalities, required and optional inputs, and generated output for each tool. CRISPRitz has three required inputs: (i) PAM sequence, (ii) a list of guides and (iii) a reference genome (in FASTA format). A collection of variants (in VCF format) and/or genomic annotations (in BED format) can be included as optional inputs. CRISPRitz performs the off-target site search by supporting a user-specified number of mismatches and/or DNA/RNA bulges. Importantly, the parallelism capability of multi-core architectures is utilized for all the computational-intensive tasks, such as search and index-genome, in order to minimize execution time. The five tools are the following:

- Add-variants, tool necessary to enrich a reference genome with a set of genetic variants extracted from a VCF.
- Index-genome, tool necessary to create an indexed version of the genome to perform faster searches and allowing the usage of bulges.
- Search, main tool of the software package, necessary to perform off-target enumeration and extraction of sequence from the genome (reference and/or alternative).
- Annotate-results, tool necessary to enrich off-target data with functional information.
- Generate-report, tool necessary to produce a set of graphical reports summarizing the results into simpler and readable images.

Each tool will be deeply analyzed and explained in the following sections, with details about implementation, used algorithms and computational time. Including also a pre-development analysis to clarify why some choices were taken.

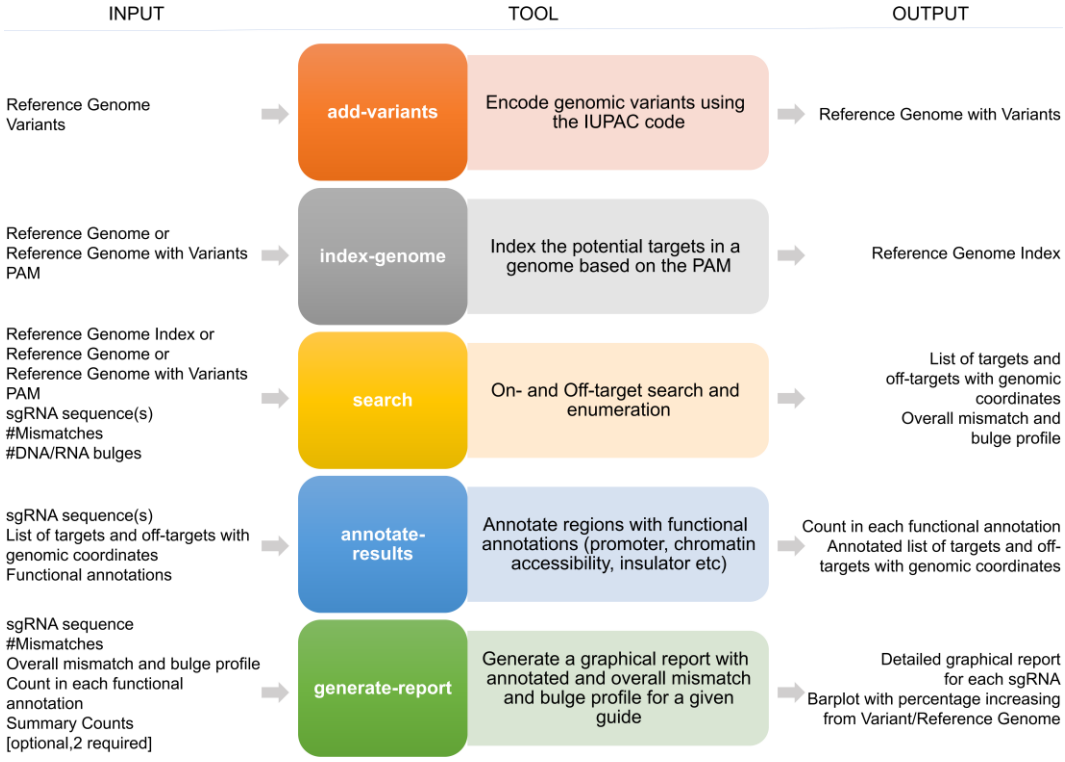


Figure 24. Overview of CRISPRitz tools.

3.2.1 Add-variants tool

This tool was developed to help a user with a complex task, such using huge dataset of genetic variants in CRISPR/Cas analysis.

This task can be accomplished in many simple but inefficient ways. The first problem that was to be solved, was the complexity of the problem itself. When dealing with many variants and many individuals from the same dataset, it's impossible to treat the problem as a common combinatorial task.

One of the first idea, was to treat each individual as a unique set of variants and generate a personal genome, containing only its specific genetic information. But it's simple to see that, if we have 22 chromosomes, each carrying a mean value of 4 million variants, with more than 2000 individuals, this solution became immediately unfeasible. In fact, applying this solution, it's equal to perform more than 2000 searches on whole genome, that even with a search of few minutes, will result in days of computation.

The second idea was to treat each variant as a subset of individuals, so taken a variant, find all the individual that share it and then generate a number of genomes equal to these subsets, trying to maximize the number of variants and individuals to put into the same genome. But also, this idea was not optimal, since we discovered that few variants are shared by more than 2/3 individuals and this approach would have led to the creation of hundreds of thousands of genomes.

Finally, we decided to take advantage of a code created by IUPAC association, known as the IUPAC nomenclature code for polymorphic nucleic acids (Johnson, 2010). This code allows to codify into a single nucleotide, any possible genetic variant. For example, if we want to codify a T and G in a single nucleotide, we will insert the letter K into the sequence and after that we will retrieve the reference and alternative nucleotide by just looking to our code dictionary, without losing any information in the process.

With this solution, it's possible to codify any variant information in a single strand reference genome, without the necessity of creating separate sequences or entire genomes for each individual or each variant to analyze. Therefore, it's possible to perform a single analysis on a genome enriched with variants, to obtain all the putative off-targets, both reference and variant induced.

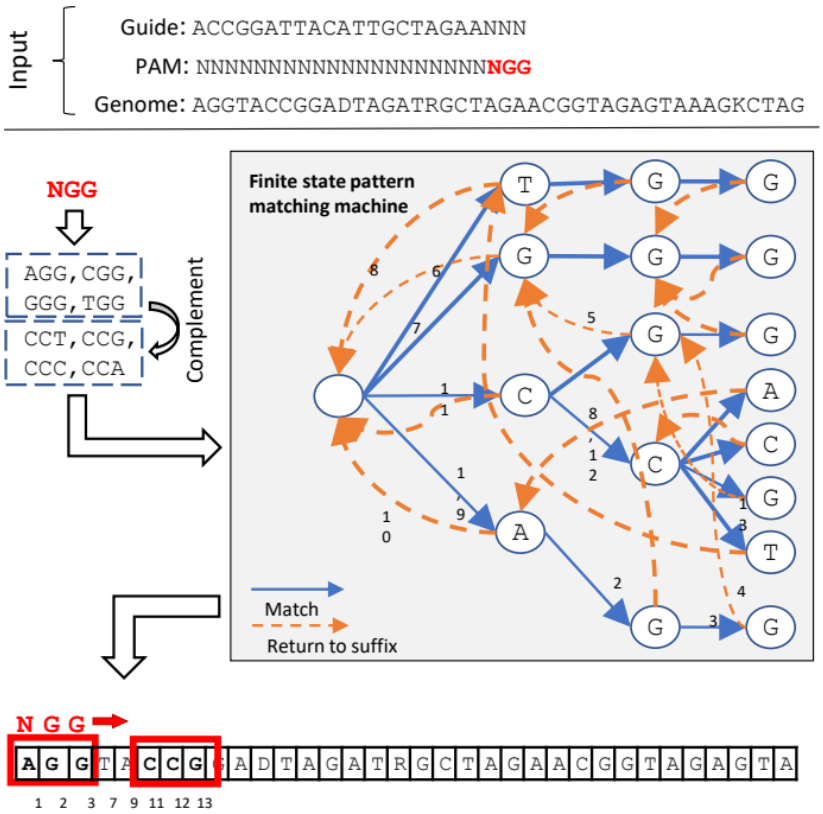


Figure 25. Aho-Corasick Automata for PAM search.

3.2.2 Index-genome tool

The two required inputs prior to genome indexing include (i) PAM sequence and (ii) a reference genome (in FASTA format).

CRISPRitz then identifies and compiles all PAM-restricted sites (hereafter referred to as candidate off-target sites) within the provided genome into a genome index data structure (i.e., one genome index for each input PAM). This allows for a reduction in execution time for all subsequent searching with bulges.

The index-genome tool starts by searching for all occurrences of the user-specified PAM in the genome.

To efficiently find all the genome regions compatible with a given PAM, the algorithm creates a small deterministic automata machine used to scan and enumerate all the potential PAM matches in linear time (based on the length of the PAM).

The automata machine (see Figure 25) reduces the searching time complexity to the minimum number of comparisons *O(N)*, where *N* is the genome length (i.e., the algorithm scans each nucleotide in the genome only once). Automata machines are usually represented as extended graphs. In this work, the automata machine is a rooted directed tree (also called trie) enriched with additional connections among nodes.

Each path from the root to a leaf represents a PAM. Figure 25 shows the patterns corresponding to the NGG PAM i.e.: GGG, AGG, TGG, CGG, CCC, CCT, CCA, CCG. With this data structure, the PAM search in the genome is performed by reading one nucleotide at a time and by matching it with a node of the graph. If the nucleotide matches with one of those nodes, then the algorithm follows the corresponding path by moving one node forward in the tree and one nucleotide in the genome.

This node traversal iteratively continues as long as the current genome nucleotide matches the current node. If it is not possible to match the next available node, the algorithm jumps back to the parent node of the graph by following special failure edges (dashed lines in Figure 25) and the search continues from the parent node. The paths following the failure edges reduce the number of comparisons by allowing the search to restart from the longest common substring that could be matched at that stage. Every time the graph visit, reaches a leaf node, the algorithm saves the current index (i.e., the nucleotide position in the genome) as it may represent the starting position of a candidate off-target site.

The full list of indices of candidate off-target sites is the output of the PAM search. Of note, the indices are saved in two arrays to differentiate between positive and negative strands.

For each identified PAM sequence, the genomic sequence adjacent to the PAM with length equal to the input guide sequence is extracted and represents a candidate off-target site. The adjacent sequence is upstream or downstream as specified by the user to accommodate CRISPR nucleases available at the time of this article (e.g., Cas9 and Cas12a). All identified candidate off-target sequences are collected, sorted following a lexicographical order and encoded using the four-bit notation as shown in the table presented in. *index-genome* is based on a ternary search tree (TST) data structure (Bentley & Sedgewick, 1998b), which is optimized for approximate string search, such as utilized for text autocompletion or spell checking.

CRISPRitz uses a ternary search tree (TST) data structure (Bentley & Sedgewick, 1998b) to index genome for rapid bulge searches.

In a TST, each node represents a nucleotide and can have a maximum of three children: left, center, and right.

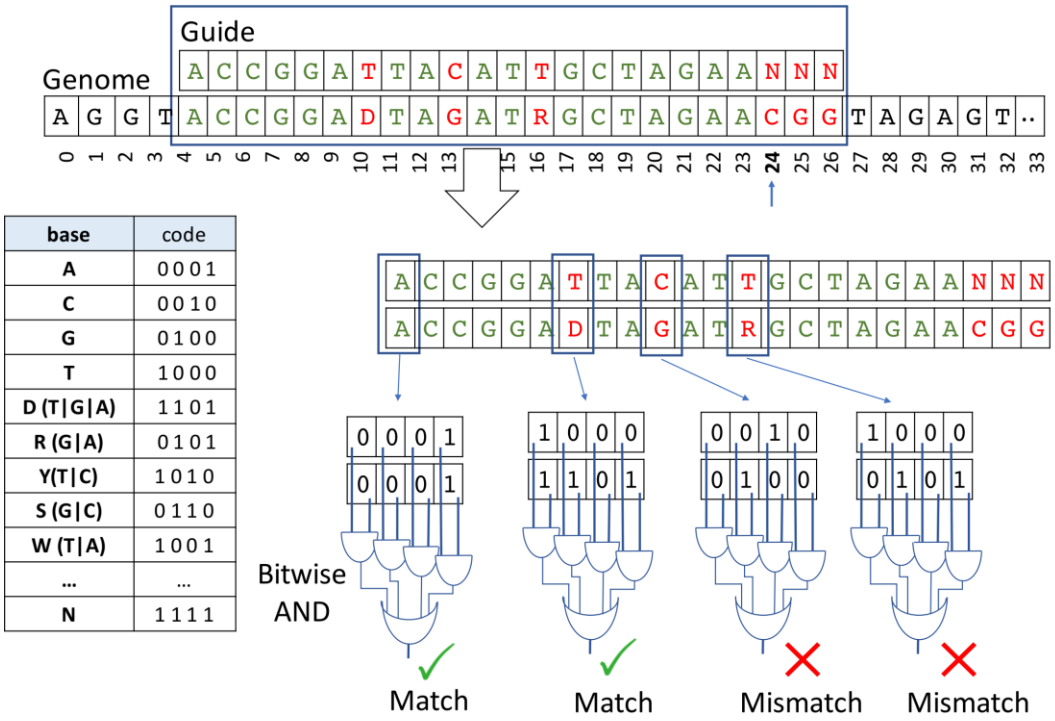
The TST is built by inserting one candidate off-target sequence at the time, where the insertion is implemented through a recursive search. Starting from the first nucleotide of the candidate target sequence and from the TST root, the algorithm compares if the two-nucleotide match, if they match, the comparison continues to the next nucleotide on the candidate target sequence and descend in the TST through the center child node and no new node is added to the TST.

If the nucleotide pair between the center child node and the nucleotide from the candidate off-target sequence do not match, then the algorithm verifies the lexicographical order of the nucleotide on the candidate target. If it is smaller than the current nucleotide in the TST node, the search recursively continues to the left child, otherwise if the candidate target nucleotide is lexicographical greater, the search recursively continues to the right child. Finally, if the node in the chosen direction does not exist, a new node is inserted in the TST with the current nucleotide of the candidate target sequence.

Every genomic sequence inserted in the TST is 2 characters longer than the PAM sequence to allow for searches with up to two DNA bulges. For example, for the PAM in Figure 25, the tool inserts a string of length 25 nucleotides (23 nucleotides for the guide plus PAM sequence as well as an additional two nucleotides for DNA bulges).

The additional nucleotides represent the appropriate PAM flanking (upstream or downstream) sequence in the reference genome. Figure 27, shows an example of TST construction, by considering three strings (candidate off-target sites) named CT1, CT2, and CT3. The algorithm first inserts CT1, which is represented (by construction) by the

central vertical path of the TST. The algorithm then inserts CT2 by visiting the first seven central nodes (corresponding to the common prefix with CT1). Since the eighth base (C) does not match with G, a new left child is created (since C < G). The procedure recursively completes the insertion by forming the left vertical path of the TST. CT3 is similarly inserted, by sharing a common prefix of length three after which a new branch is created on the right starting from T (T > A).



base	code
A	0001
C	0010
G	0100
T	1000
D (T|G|A)	1101
R (G|A)	0101
Y(T|C)	1010
S (G|C)	0110
W (T|A)	1001
...	...
N	1111

Figure 26. Brute-Force Search, with IUPAC code tables and bitwise operations.

3.2.3 Search tool

The search tool searches for candidate off-target sites with mismatches only or with both mismatches and bulges. It is necessary to use an index genome when considering both mismatches and bulges because the bulge-related search is computational expensive, and the index genome is fundamental to obtain a robust reduction in computational time by pre-computing an efficient data structure to perform searches.

The search operation is a computationally intensive task largely due to the required number of base-to-base comparisons. However, CRISPRitz implements this task through exploiting the fast bitwise comparison of the four-bit code used to represent each nucleotide in the genome, generated using the IUPAC code. The inputs for analysis include:

i. Arrays of indices generated by the PAM search
ii. Input list of guides
iii. User-specified maximum number of tolerated mismatches and bulges
iv. Reference/Enriched Genome (reference genome with variants)

If a given guide from the input guide list matches to a region of the genome without mismatches, the index (or the indices) represents the starting position of one (or more)

on-target site(s), otherwise the reported position is referred to as a candidate off-target site.

CRISPRitz also implements a search algorithm that handles a user specified number of mismatches and DNA/RNA bulges. This algorithm requires the construction of an index genome as presented in Section 3.1.3.

Usage of an indexed genome allows the software to search on the whole genome in $O(\log(n) + k)$ with k as length of the searched sequence and n, number of putative off-target sites.

This search is implemented with a function that recursively visits the TST. For all the candidate off-target sites, the algorithm begins from the TST root and visits all TST branches by checking if the nucleotide comparison corresponds to a match, a mismatch, or a bulge (DNA/RNA).

When the user allows for bulges, then CRISPRitz can return duplicate results. For example, the results may represent the same target with a different number of mismatches and/or bulges or in different positions.

In the mismatch-bulge search type, the visit on the branch can reach the leaf if the bulge threshold is sufficient to visit all the nodes in the branch. This may happen since the DNA bulges are treated like supplementary characters on the guide that can match with supplementary characters on the candidate off-target site.

The mismatch-bulge search type stops when a branch of the TST is visited completely (i.e., when the leaf is reached), or when the TST is visited partially and any threshold of mismatches or bulges has been exceeded, or when the visit has reached a number of nodes equal to the guide length. The candidate off-target site search is a function that recursively visits the TST, in a similar manner as during the TST construction, described in the previous section.

With the constructed TST, the search is performed in two different ways, the first only considers mismatches (hereafter referred to as 'mismatch search'), and the second considers mismatches in which bulges are permitted (hereafter referred to as 'mismatch-bulge-search').

In the mismatch-search type, it is not possible to reach any leaf of the TST by construction (see Section 3.1.3) and every candidate off-target site is written only once (a candidate off-target site cannot be saved, for example, with 2 mismatches and with 4 mismatches considering the same guide and the same genomic position).

The mismatch search type stops either when the pattern (the candidate off-target site) has been found or when the number of maximum mismatches has been exceeded (the pattern, or candidate off-target site, is not reported).

In the mismatch bulge-search type, mismatches and bulges are permitted for candidate off-target sites (see Figure 27).

To avoid computationally expensive search iterations on the same branch, the algorithm implements a recursive approach that tries all possible combinations of results a candidate off-target site can generate.

Those combinations are tested recursively on the TST branch at runtime, avoiding an iterative search restarting every time from the TST root. For example, if the target guide is CCCAACCC, two possible combinations with bulges that share trace-backs in the recursion are CCCA-ACCC and CCCAACCC, since they share the common prefix CCCA. Every time an off-target site has been saved, the algorithm computes a recursive trace-back and tests further combination of mismatches and bulges on the same branch.

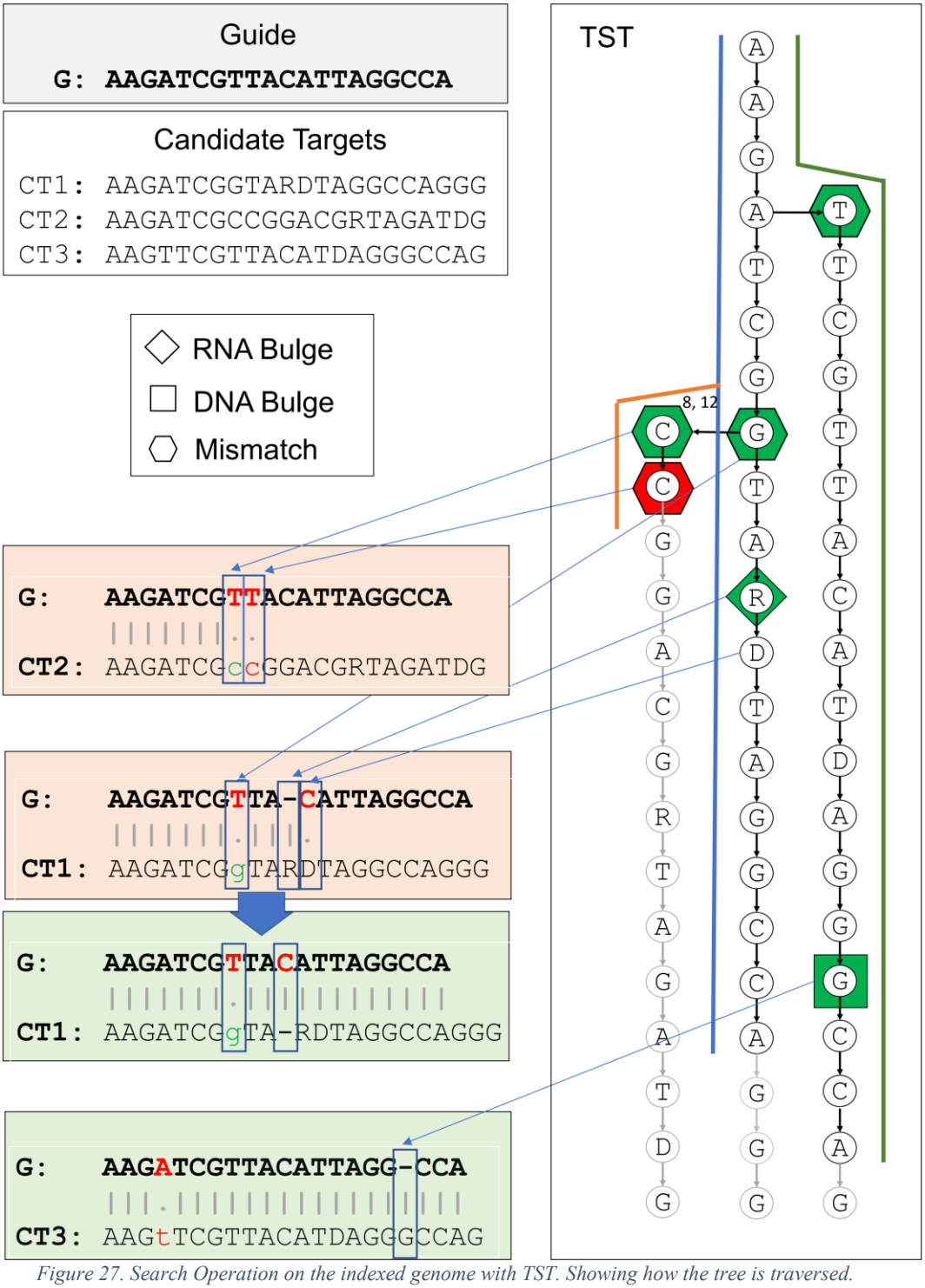


Figure 27. Search Operation on the indexed genome with TST. Showing how the tree is traversed.

3.2.4 Annotate-results tool

CRISPRitz allows the user to annotate genetic regions with functional data.

The software takes in input a list of genetic regions with a specific function (e.g., exon, intron, promoter, CTCF, etc.) and annotates each putative target using this information.

The annotation is done by using an interval-tree structure (Chaim, 2014). The structure is created by inserting the start and end coordinate of each genomic region present in a text file.

The function takes in input a file containing a set of genomic regions with start and end coordinates, plus the genetic function assigned to that region.

Then, CRISPRitz generates the interval-tree reading all the region's coordinates and generating a balanced tree, allowing for fast searches.

When a putative off-target region, falls into an interval of the tree, the interval is extracted from the tree and the target is annotated with the correlated function to that specific region.

With this function the user can provide any functional annotation data and the software can mark each target with the function provided by the user.

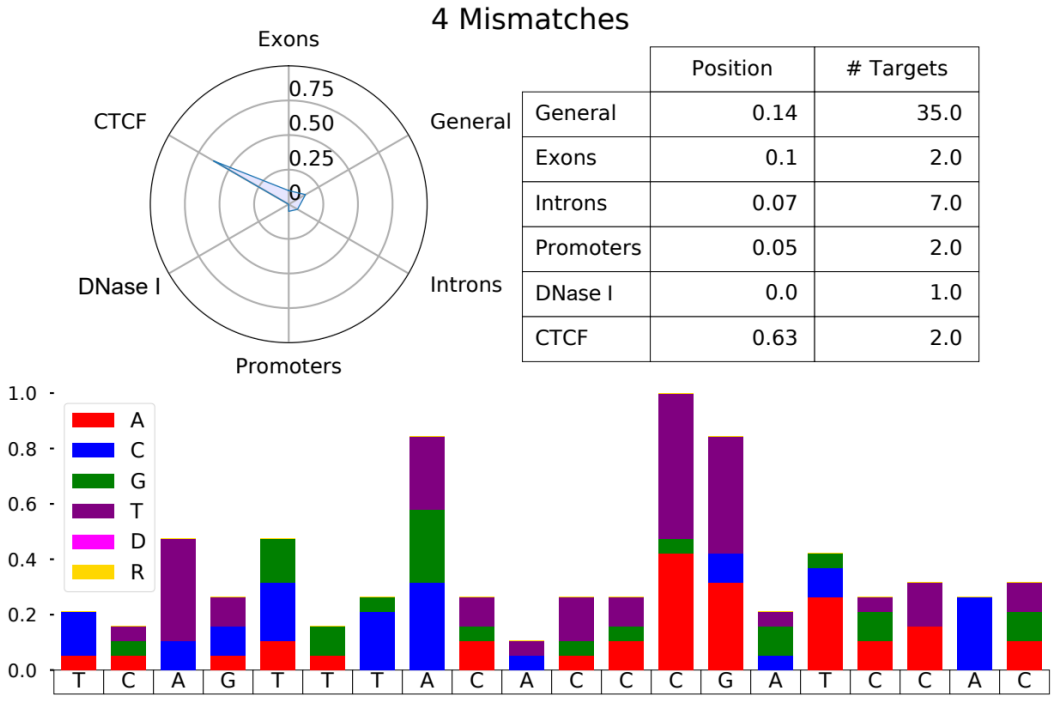


Figure 28. Example of report with annotated targets and motif logo

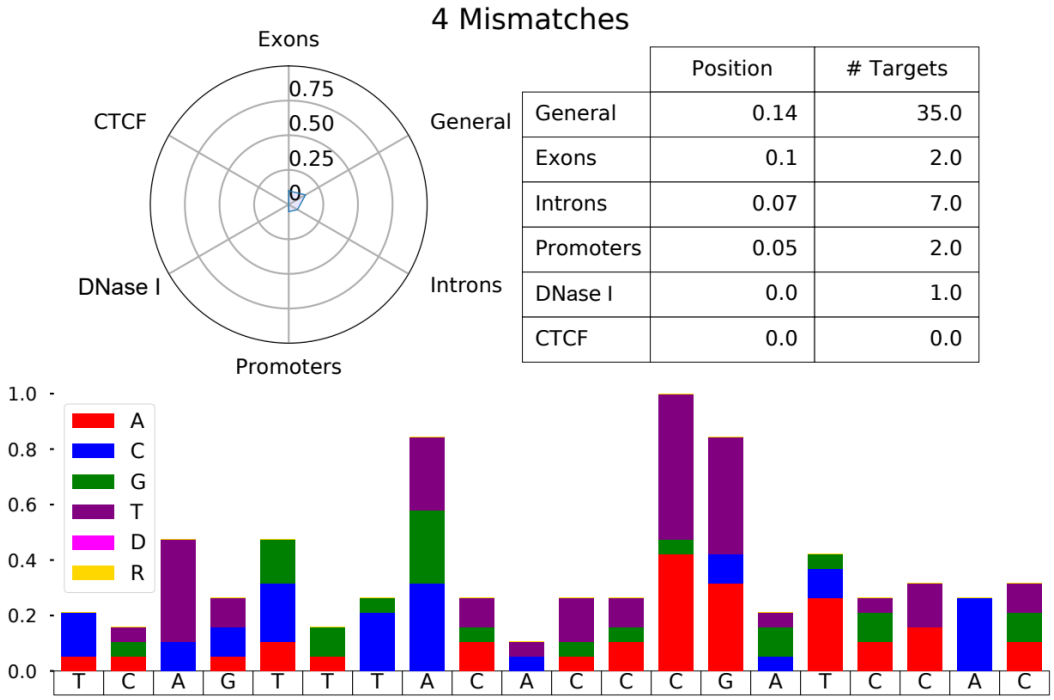


Figure 29. Example of report with annotated targets and motif logo

3.2.5 Generate report tool

Based on the annotated results, generate-report creates a graphical report to aid in the assessment of the potential functional impact of the off-target sites for a given guide.

Specifically, a radar chart is created for each mismatch threshold.

Each axis of the radar chart corresponds to a functional annotation and the plotted value represents the similarity (in terms of found on-/off-target sites) to the examined guide as compared with its own guide set (so the user can assess an individual guide's behavior as compared with all other guides from the input guide list) or to a previously analyzed guide library (e.g., the Gecko Library v2; (Sanjana et al., 2014)).

The area in the radar chart (see Figure 28, Figure 29, Figure 30) allows the user to quickly evaluate the off-target potential for a given guide. A small area on the radar chart corresponds to a guide with reduced candidate off-target sites (the exact number is displayed for each annotation) whereas a large area corresponds to guides with increased candidate off-target sites in multiple functional regions.

generate-report also produces mismatch profiles where each position corresponds to a bar that represents the number of observed mismatches for each nucleotide normalized on the maximum number of mismatches. If genetic variants and enriched genomes are used during the search operation, generate-report can also plot a bar plot showing the percentage gain for each annotation and the additional sites as compared with the reference genome using the annotation file created by annotate-results.

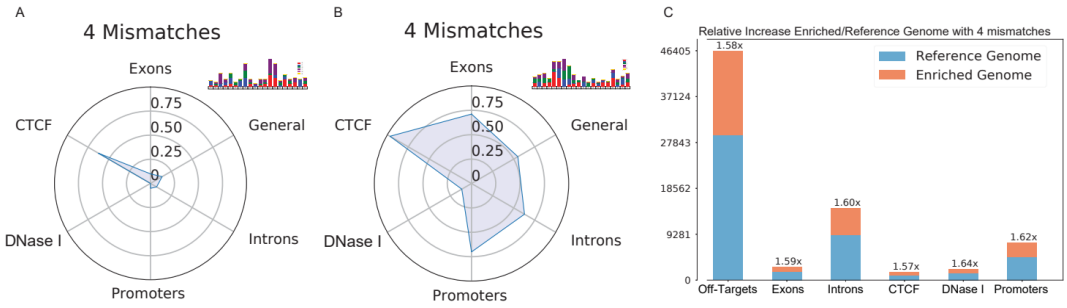


Figure 30. Set of charts generated by CRISPRitz, containing also annotations.

3.3 CRISPRme, web-based tool to analyze variant induced off-targets

CRISPRme was developed to analyze the complex and difficult outcomes of variant generated off-targets. CRISPR/Cas off-target enumeration is per se, a difficult and non-trivial task, since it's easy to produce lots and lots of putative off-targets just by searching sequences on the genome, but it's not easy to generate significant reports and panels from these huge number of targets. Furthermore, the introduction of variants makes the task more complex, since the tool needs to consider many more targets and different conditions, such as the new haplotypes introduced by variants.

When the development of CRISPRme started, no other software was able to complete this task in a reasonable amount of time and with the completeness and the accuracy of CRISPRme.

CRISPRme considers many factor that are fundamental to properly complete this analysis. First, it takes into account the individual information, analyze each target and extrapolating the sample and the variant(s) present in it, this accuracy is fundamental to give the user a specific and precise view on the final results. Furthermore, it takes into consideration the real haplotypes generated by the variant data. If a phased VCF is used during the creation of the alternative genome, the software changes the analysis to consider phased information and reports only real haplotypes, discarding the artificial haplotypes generating by the collapsing of the whole variant information into a single genome.

Lastly, the software generates many graphical and detailed written reports, necessary to better understand how the variants induced off-targets are changing the outcomes and to define how incisive these were in changing the results.

All of these, will be explained in the following sections, starting with a general explanation of CRISPRme implementation and continuing with more details on the alternative targets processing and accounting.

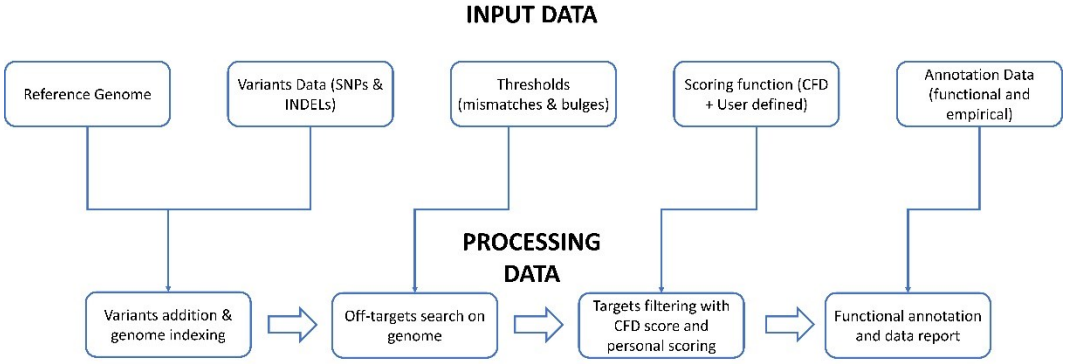


Figure 31. CRISPRme workflow with input and process.

3.3.1 CRISPRme general implementation and functions

CRISPRme was developed in collaboration with clinicians and lab technicians. This collaboration was fundamental to abandon the more technical and informatics approach, to focus into a more user-oriented software, easy to use, easy to understand and with a task-oriented implementation rather than a complex and cryptic implementation for a non-informatic user.

With this objective in mind, the development was followed during each phase by comments and suggestions from our collaborators, to always maintain the task-oriented approach.

CRISPRme was developed around CRISPRitz, in fact, the search phase, the index phases and the alternative genome creation, are demanded to CRISPRitz.

CRISPRme was developed to use CRISPRitz raw results and analyze them, performing detailed analysis of the alternative targets and to collect and report more specific information about targets distribution in terms of mismatches and bulges, in terms of score and how the different populations and individual's variants, are accountable for changes in the final outcomes in the sgRNA enumeration.

The main function of CRISPRme starts when CRISPRitz results are generated. These functions can be split into different routines:

1. Separation of reference and alternative targets, this routine is fundamental to gain information about the origin of the target, if the putative off-target is found in the reference genome or if a variant is introducing a new putative off-target, not present in the reference genome.
2. Conversion of variants in IUPAC coded targets and scoring, this routine is the core of CRISPRme, since it's the function that converts all the IUPAC nucleotide from targets into real nucleotide and verify if the haplotype is real, so belongs to a set of individuals, or it's just an artificial haplotype resulting from the collapse of the whole variants set into a single genome. This routine also completes the scoring for the targets, calculating scores only for real haplotypes.
3. Target merging by position, this routine takes into care all the possible targets falling into nearby regions, avoiding the over representation of a single region due to repetitive sequences present in the DNA.
4. Graphics generation and reporting, this routine generates all the graphical reports, including score graphs, summary reports and populations/individuals' plots. It also generates the whole reports package, like a general mismatches and bulges table, a summary matrix that groups by targets for mismatches and bulges count and a table that is grouping targets by individual that shares the same target.
5. Integration and panel creation, this routine is the final routine of CRISPRme. It generates the integrated file for the user, this file is fundamental since is the file that is subject to all the queries that the website provides to the user. It is used to calculate the summary and generate the graphs and the graphical reports. Furthermore, it collects all the targets adding also a specific ENCODE (E. P. Consortium & others, 2004) and GENCODE (Frankish et al., 2019)(Frankish et al., 2019) annotation, very useful to understand which gene is nearest to the target, and which genetic function the region is reported to do. In the end, the integrated file can be used as a screening panel since the targets are filtered, corrected and unique.

All these routines are internally managed by CRISPRme, and the user needs only to wait and collects the results. Everything is stored and saved into a run-time generated

directory, and everything can be seen and searches directly from the web-based GUI or using the integrated targets file.

3.3.2 Determination of putative off-targets origin

Selecting targets depending on their origin, is one of the main tasks done by CRISPRme.

Since it is fundamental to inform the user if the reported target is present in the reference genome or is induced by a variant.

This task is done by comparing results obtained from a search on the reference genome with results obtain from searches done with alternative genomes (the process can account for more than one alternative genome).

The software, groups the target per chromosome and position and then determines if a target is already present in the reference or not, by applying a flag to all non-reference targets found.

The process is done by using a combination of python scripts and Linux bash functions.

The used Linux functions are mainly sort operations to sort in the correct order the targets. Then the sorted targets are input into a python script that analyze if the targets into a specific genetic region exist in reference and also in alternative genome.

If a target is found into both genomes, nothing is done.

In the case a target is variant-induced, the targets itself is flagged with a key.

With this easy and fast process, any target is flagged with its origin and in any further processing, including the final report, it is possible for the user to determine the origin of the targets just by looking at the presence of the flag.

3.3.3 Off-targets selection and IUPAC code conversion

This task is the main and more complex operation done by CRISPRme, the core of the software.

The task starts by processing all the off targets found during the search, reference targets are scored, as explained in the Section 3.3.4, and returned to be saved in a result file.

Variant-induced targets are subjected to a pre-scoring phase, in this phase the target is analyzed and any present IUPAC nucleotide, is converted to a real nucleotide, generating so a set of nearly identical targets with single nucleotide polymorphisms.

This process is very computational demanding, since testing all the possible combinations of codified nucleotides in the IUPAC, will result in an exponential generation of targets.

For example, the following target,

AATCTCACTKACYACRATCATGA

Can be converted into 8 different sequences:

1. AATCTCACTGACCACAATCATGA
2. AATCTCACTGACCACGATCATGA
3. AATCTCACTGACTACAATCATGA
4. AATCTCACTGACTACGATCATGA
5. AATCTCACTTACCACAATCATGA
6. AATCTCACTTACCACGATCATGA

7. AATCTCACTTACTACAATCATGA
8. AATCTCACTTACTACGATCATGA

This simple example clarifies that the number of possible combinations is the multiplication of the cardinality of nucleotides codified into a single IUPAC nucleotide.
In the example,

- K can be converted into G and T
- Y can be converted into C and T
- R can be converted into A and G

So, if the IUPAC nucleotides are in a set $I = \{K, Y, R\}$ and Ii is the i-th IUPAC nucleotide, the count of resulting targets is $\prod_1^n |Ii|$, where n is the number of IUPAC nucleotides present in the original target.
Thus, resulting into a computational time of $O(m^n)$, where m is the cardinality of the IUPAC nucleotide processed and n is the count of IUPAC nucleotides present in the target.
With limited number of IUPAC nucleotides in the same target, the operation remains feasible in acceptable time, but with some cases, for example targets with 10, 15 or 20 IUPAC nucleotides, the time to process even a single targets become too much.
So, we analyzed the targets and discover that many combinations are non-real, they are artifacts derived by the introduction of a fake genotyping profile due to collapsing all the variant into a single sequence.
After this discovery, we completely rewrite the code processing the variant targets.
Now the process is the following:

1. Extraction of IUPAC nucleotides from the variant sequence.
2. For each extracted IUPAC nucleotide, account the samples carrying the mutation.
3. Starting from the first encountered IUPAC nucleotide, try the combination of IUPAC nucleotides that shares at least one sample, keep the target, discard the others.
4. Processing the target from start to end with these criteria, so selecting a fixed nucleotide and try to generate a real resulting target.
5. Repeat for each nucleotide, avoid repeating the process for already tested combinations.

An example of this process, utilizing the preceding example.

In the target,

AATCTCACTKACYACRATCATGA

There are 3 IUPAC nucleotides: K, Y, R.
Each nucleotide belongs to a set of individuals, for example,

K = {sample1, sample2} → G, T
Y = {sample1} → C, T
R = {sample3} → A, G

For each IUPAC nucleotide, one coded nucleotide (blue) is the reference and the other (red) is the alternative.

Processing the target, the software selects the first nucleotide as fixed position of analysis, so starting from K, it substitutes the IUPAC code nucleotide with T and extract the sample group carrying the variant nucleotide {sample1, sample2}. Then, the target is read in its entirety and, when a IUPAC nucleotide is found, is tested against the set of samples carrying the fixed variant nucleotide.

So, in this example case, when the Y is encountered, two paths are taken by the software. The first one, converts the nucleotide in the reference allele (blue) and the process continues, the second one, converts the IUPAC nucleotide into the alternative allele T and intersect the set of samples from the fixed nucleotide with the set of samples $\{sample1\}$ of this alternative nucleotide $\{sample1,\ sample2\} \cap \{sample1\} = \{sample1\}$. The result is a set containing $\{sample1\}$, since the set is non-void, the process can continue from this position to the end of the sequence. In this case, since the last variant nucleotide does not share a sample with the current set {sample1}, the target is saved with the two alternative nucleotides, for the following phase.

This process is repeated starting from the second encountered IUPAC nucleotide, Y.

Now this nucleotide is treated as the fixed point, and any preceding combination is not evaluated since was evaluated before.

After all the possible combinations are generated, only the target with a non-void set of samples is kept, the targets with void sets of samples, represent non-real targets generated by some artificial combination of reference and alternative nucleotides are discarded and not processed.

Applying this algorithm, the process become executed in quasi-linear time $O(n * m)$ because when a target with more than one IUPAC nucleotide is processed, each time a new nucleotide is converted, it is also checked, at run-time, if the set of samples carrying the nucleotide is a non-void set when intersected with the fixed point nucleotide and if the resulting set is void, the process is stopped and no more combinations are generated, avoid the generation and processing of non-real targets.

When the process is complete, the set of generated targets are scored, as explained in the following section, and then returned to the main function to be written into a result file for further processing.

3.3.4 Derived targets scoring function

To score targets using Cutting Frequency Determination (CFD) (Doench et al., 2016b), CRISPRme uses a matrix generated by the authors who introduced the scoring system based on empirical data.

This matrix is composed of all the possible pairs of mismatches between an RNA and DNA sequence with a length of 20 nucleotides.

Each entry in the matrix file reports an RNA and DNA nucleotide pairing. For example, the entry “rA:dG,20', F0.227” means that the RNA nucleotide A paired with the DNA nucleotide G in position 20 will have a score of 0.227.

This value is multiplied with values for any other mismatches present to obtain the CFD score for the off-target sequence. If a sequence has only 1 mismatch, its final score will be the score of the mismatch, so in the previous example, the CFD score of an off target with only one mismatch (20A>G) will be 0.227.

The matrix also contains scores for bulges, which are indicated as “-”. An example entry representing a bulge is “sS'r-:dA,2', F0.692,”which indicates that a RNA bulge pairing with the DNA nucleotide A at position 2 has a score of 0.692.

Bulges are not allowed in the first position of the RNA or DNA sequence.

Computationally, targets with DNA bulges are reported with a longer sequence with respect to the original spacer. To avoid inconsistencies when calculating CFD score for

off-targets with DNA bulges, we calculate the score based on only the last 20 nucleotides of the protospacer as intended by the original CFD scoring method.

The process also scores the PAM nucleotides. Since CFD score was derived from SpCas9 data, the matrix only contains scores for NGG and all the possible combinations of the last two positions of the PAM.

An example of an off target with a DNA bulge:

Spacer: CTAACAGTTGCTT-TTATCACNNN

Protospacer: CTAACAGcTGCTTCTTATCACCTC

This off-target contains one mismatch (in lowercase) and one DNA bulge (aligned with gap in the spacer sequence). When we calculate its CFD score, we do not consider the first nucleotide of the protospacer because the protospacer and the spacer are one nucleotide longer than the usual 23 bps scored by CFD.

Then each mismatch and bulge are scored according to the matrix and the value of each pair is multiplied together to obtain the final CFD score.

In this example, the first mismatch is rT:dC in position 7 (we skip the first nucleotide since there is 1 DNA bulge), and the score for that pair is 0.588. Then, we encounter the bulge r-:dC in position 13, and the score of this pair is 0.

We keep moving along the sequence until we reach the PAM. In this case, the last nucleotide couple TC has a score of 0. Finally, we multiply each value saved during the process, so the final score is calculated as 0.588 * 0 * 0, yielding an off-target CFD score of 0 for this example.

Scoring off-targets with RNA bulges is simpler since the sequences are not elongated due to the bulges. An example:

Spacer: CTAACAGTTGCTTTTATCACNNN

Protospacer: CTAACAGTTGCTTTTAT-ACGTG

This off-target only contains an RNA bulge (protospacer gap). We scan the sequence and encounter the bulge in position 18, so we use matrix entry “rC:d-,18” with score 0. The final CFD score for the off target will be 0.

In addition, a global score called CFD specificity score is provided for each guide and defined as follows:

$$CFD\ specificity\ score = 100 * 100/(100 + \sum_{target_i} CFD(target_i))$$

where $target_i$ is one of the enumerated off-targets from the search.
Its range is (0, 100], where a gRNA with no predicted off-targets given the search parameters has a CFD specificity score of 100.
Of note, the CFD score is only applicable for SpCas9. CRISPRme does not calculate CFD scores when searching for editors other than SpCas9. In this case, a -1 value is reported for each off target.

3.3.5 Merging targets in same genetic positions to avoid over representation in final results

This task was required specifically by our collaborators.

After the first testing phase, one of the most annoying things reported by our collaborators, was the over-representation of certain genetic regions when account for off-targets. In fact, when using bulges, the raw results reported by CRISPRitz and

processed in CRISPRme, contains a lot of ‘duplicates’ targets, so targets with a different combination of mismatches and bulges, that fall in the same genetic region.

This is caused by the combinatorial nature of the bulge representation in the informatic processing of the problem.

Another problem, reported by our collaborators was that near targets were treated as different targets, so if a target is found in position *X* and the following targets is found in position *X+1* they represent the same genetic region, biologically speaking, but are treated as completely independent from a computational point of view.

To solve this problem, we discussed with the collaborators to better understand how produce useful and intelligible result without compromising the readability and the completeness of the data.

The answer was, preserving the highest scoring target (or the target with lowest sum of mismatches and bulges) in a determined genetic region plus a small window of nucleotides.

So, to accomplish this task, without losing data and returning a comprehensive list of targets, CRISPRme takes the list of targets from the preceding scoring phase and with sorting operations, returns a list of targets ordered by chromosome, position and score (or sum of mismatches and bulges).

After this sorting operation, done with the efficient Linux implementation of merge sort (Bron, 1972), the list of targets is processed by selecting all the targets in a specific position and all the targets in a user defined window (3 nucleotides by default), creating a cluster of ‘near’ targets.

This cluster is then processed to preserve only the highest scoring (or lowest mismatches and bulges count) target.

This is accomplished by keeping only the target with the characteristic and discarding the others. The discarded targets are then saved into a ‘discarded’ file to keep track of any other possible valid target for the specific genetic region.

The top target is then saved and reported in the final result file.

With this operation, we generate, for the final user, a simpler and easy to read file, avoiding the over-representation of certain region without losing any information and possible hazardous target.

3.3.6 Graphical report generation and summary reports

In this phase, CRISPRme process the final list of targets, polished and refined to be given to the user.

After all the processing operations, the targets are used to generate many summary reports, fundamental to easy visualize the search results and better understand the general distribution of the targets, like the count of targets in different combinations of mismatches and bulges, or how many targets belong to a specific population.

These tables and matrices are fundamental for the final user to understand, without looking at the exhaustive target data, if a guide has many targets in low count cells or for example if a specific population is more afflicted by putative off-target.

gRNA (spacer+PAM)	Nuclease	CFD specificity score (0-100)		Total	# Bulges	0MM	1MM	2MM	3MM	4MM	5MM	6MM
filter data...												
				109844	0	1	0	7	148	1626	13634	94428
			REFERENCE	3708218	1	1	13	477	6953	72849	535709	3092216
				43025008	2	14	409	9088	119916	1099215	7190525	34605841
CTAACAGTTGCTTTTATCACNNN	SpCas9	0.417										
				8371	0	0	0	2	16	159	1177	7017
			VARIANT	441837	1	0	4	39	521	3984	27388	409901
				12940187	2	2	37	416	4611	38883	293277	12602961

Figure 32. General table summarizing results. In the first three columns, are reported guide characteristics, like the gRNA sequence, the nuclease, the CFD specificity core. The last columns, represents the count of targets falling in the respective category (sum of mismatches and bulges).

Custom Ranking | Summary by Mismatches/Bulges | Summary by Sample | Query Genomic Region | Graphical Reports | Personal Risk Cards

Focus on: CTAACAGTTGCTTTTATCACNNN

Summary table counting the number of targets found in the Reference and Variant Genome for each combination of Bulge Type, Bulge Size and Mismatch. Select 'Show Targets' to view the corresponding list of targets.

Bulge type	Mismatches	Bulge Size	Targets found in Genome			PAM Creation	
			Reference	Variant	Combined		
X	0	0	1	0	1	0	Show Targets
RNA	0	1	1	0	1	0	Show Targets
DNA	0	2	2	0	2	0	Show Targets
RNA	0	2	12	2	14	0	Show Targets

Figure 33. Mismatches/Bulges matrix, reporting a more complete view of the general table. The targets are accessible by clicking the link 'Show Targets' on the right.

Custom Ranking | Summary by Mismatches/Bulges | Summary by Sample | Query Genomic Region | Graphical Reports | Personal Risk Cards

Focus on: CTAACAGTTGCTTTTATCACNNN

Summary table counting the number of targets found in the Variant Genome for each sample. Filter the table by selecting the Population or Superpopulation desired from the dropdowns.

Select a S.. | Oroqen × | Select a Sample | FILTER

Download file

Sample	Sex	Population	Super Population	Targets in Variant	Targets in Population	Targets in Super Population	PAM Creation	
HGDP01212	female	Oroqen	EAS	381343	813078	3674135	0	Show Targets
HGDP01209	female	Oroqen	EAS	381075	813078	3674135	0	Show Targets
HGDP01205	male	Oroqen	EAS	378893	813078	3674135	0	Show Targets
HGDP01204	male	Oroqen	EAS	378598	813078	3674135	0	Show Targets

Figure 34. Matrix representation of targets grouped by Population of samples carrying them. In the figure are also counted the number of targets in each Super-Population.

In this phase, CRISPRme produces also the graphical reports and the necessary files to generate run-time reports when the user required them from the GUI.

The produced reports, contains a visual representation of distribution of targets in a cartesian plot where each target is represented with two scores. Each target is reported with a reference score and alternative score. The first score refers to the score result for the sequence extracted from the reference genome. If a target has no variant in it, its reference and alternative score, are identical, since the target is identical in each representation.

If a target is alternative, the scores reported are two, one for the reference sequence and one for the alternative sequence, since a target is preserved only if it's score is the highest one, if an alternative target is preserved in the final representation, means that any reference target in the same region, was discarded.

In this representation the targets are drawn as points, where the point dimension is the MAF (Danecek et al., 2011) of each alternative target (MAF=1 for reference targets). Then the points are connected to represent the increase in score induced by variant addition in reference targets (see Figure 35).

With this simple, yet effective representation, it's possibile for the user to understand how variants altered the results with respect to the solo reference analysis.

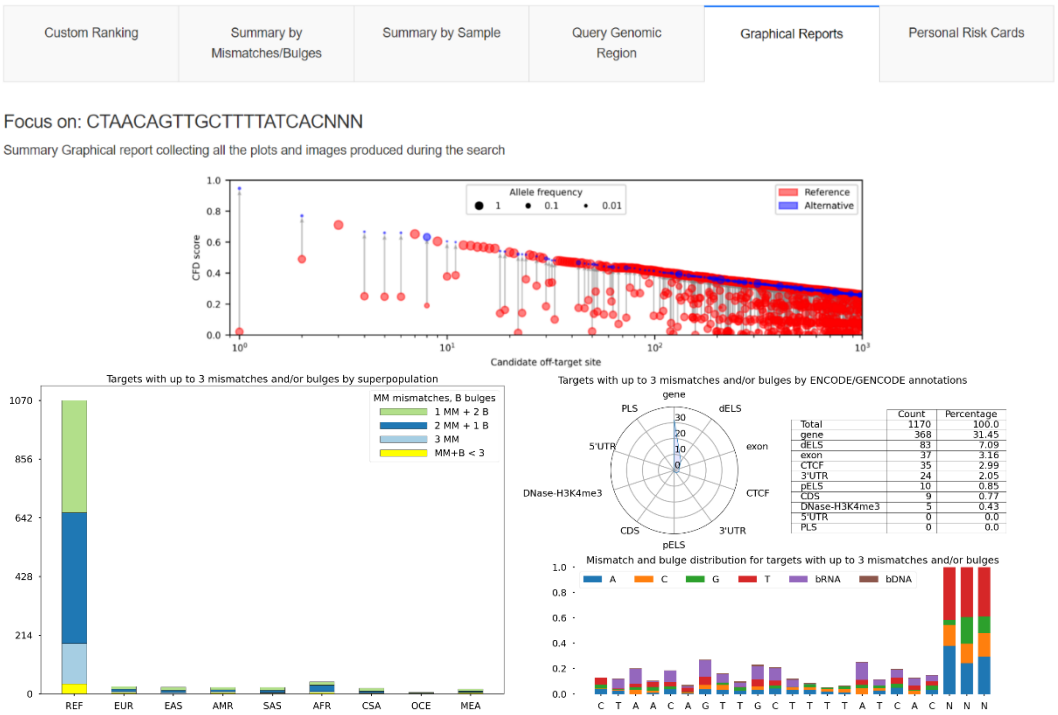


Figure 35. Graphical report generated.

3.3.7 Targets file integration, filtering and polishing

This final task is done by CRISRPme after all the other operations.

Since we want to have a unique and comprehensive file, this operation needs to be done after all the procedures are completed.

This file contains all the targets selected during the previous filtering phase, in which reference and alternative targets are merged per near positions and only the highest scoring (or with lowest count of mismatches and bulges) target is reported.

In this phase, each target is processed to be annotated with the nearest gene found using the start and ending positions of the sequence, the annotation is made by using the GENCODE (Frankish et al., 2019) dataset. Paired with the functional annotations extracted from ENCODE (E. P. Consortium & others, 2004) and GENCODE, the user can track where the target is on the genome, and how it's presence can possibly interact with the functionality of the gene itself. This information is very important to predict any possible harmful and dangerous outcome.

During this process, for each target are calculated the count of mismatches and bulges in the seed and non-seed part of its aligned sequence.

Seed and non-seed elements of the target are the half sequence nearest to the PAM and the half sequence furthest from the PAM, respectively.

These elements are important to track, since the presence of mismatches and bulges in different positions, can affect the likelihood of the cut itself and become an important element when predicting if a target will be found or not in a wet-lab experiment.

One last important operation done during this phase, is the calculation of the relative position of any variant present in the target, with respect to the target itself.

In fact, when a variant is incorporated to the target, it's absolute position, extracted directly from the VCF dataset, can be misleading in identify the correct nucleotide or group of nucleotides in case of an INDEL, interested by the mutation.

The targets are subjected to different repositioning due to strand inversion, elongations due to bulges and insertions/deletions caused by the variants. Due to this, it is necessary to recalculate the relative position for each variant present in the target, to help the user understand exactly what happened on the sequence.

This procedure takes the alternative sequence and the reference sequence, both aligned in the same way on the sgRNA, and compares them, nucleotide by nucleotide, to track where the first different nucleotide is introduced. This position is then reported into a string, containing also the reference and alternative nucleotide, in this format:

A12G

Where the first nucleotide, in this case, A, is the reference nucleotide, the position 12, is the relative position with respect to the target, so the 12 nucleotide (including any possible bulges) and the alternative nucleotide, G.

With this compact but effective visualization, the user can easily track each variant present in the target.

When the file is completely processed, the last routine of CRISPRme is started.

This routine processes the integrated file and creates an SQL database (Allen & Owens, 2010), with this database is possible to speed up any following search on the file and do complex query faster than any other method.

This file is used for all the queries and visualization available on the website and can be used by any technician to perform direct extraction in a faster and secure way.

3.3.8 Graphical User Interface for off-target assessment and analysis with graphical support

CRISPRme was developed keeping in mind that usability and ease to understand the results, were the main aims of the project.

The graphical user interface was developed with this in mind.

The software comes with two central pages, the first one is the main submission page (see Figure 36), the page that takes the user input and offers the possibility to select different analysis operations and input.

In this page, the user can input a set of sgRNAs, select a PAM sequence and a nuclease from a set, then select the reference genome, the set of variants to include and finally if the software should or not annotate the results with genetic functions and gene proximity. Using the online version, the software also allows the user to receive a completion mail when the analysis is terminated.

This submission page was developed following the suggestions and comments from our non-informaticians collaborators, so it's completely structured and designed to be user friendly and simple to understand.

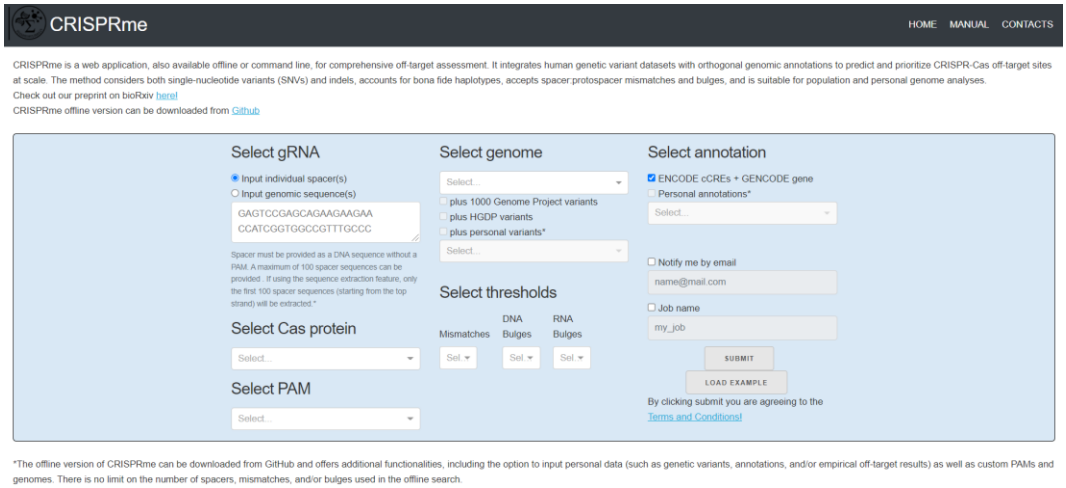


Figure 36. Main submission page of CRISPRme.

The second central page of the software is the result page.

This page allows the user to analyzed and query the results in an easy and intuitive way.

The page is composed by two halves, the first half, is the general summary table, reporting a comprehensive count of targets divided by mismatches and bulges count, basically a matrix in which each cell is reporting the count of targets that have *N* mismatches and *K* bulges (DNA and RNA) (e.g., 300 targets with 3 mismatches and 1 bulge).

The second half of the page, it's the interactive part of the page, in which the user can read results and query the database of targets.

In this half, there is a tab bar, showing the 6 tabs available to the user, to query the results in different ways and collecting all the graphics generated by the website.

The tabs are the following (see Figure 37, Figure 38, Figure 40, Figure 41, Figure 42, Figure 43):

- Custom ranking tab, the tab allows the user to perform column wise queries on the targets databases, returning a data-table containing the results. In this tab the user can perform different pre-set query, like filtering targets with the highest CFD score or with the lowest count of mismatches and bulges. Then, it's possible to select two columns on which perform group by and sorting. For example, it's possible to show the set of targets with highest CFD score, then group them by mismatches count and score, select a minimal and maximal value to show and then order them in descending order.
  These pre-set options are the common searches a technical user will perform, according to our collaborators that guided this page development.
- Summary by Mismatches/Bulges tab, the tab allows the user to account for specific subgroups of targets, grouped by mismatches and bulges count divided into specific bins. The resulting matrix collects all the targets that falls into a specific category, for example, targets with 1 DNA bulge and 3 mismatches. In this tab, the user can easily access these targets, by pressing the 'show targets' button and be redirected to a data-table, containing the requested results.
- Summary by Sample tab, the tab contains the information for each individual processed in the search, such as, the sex, the population and superpopulation (e.g., AMR, AFR, etc.…). Each row reports the associated number of targets that shares the sample, counting also the targets in the population and in the superpopulation. This page can be filtered by the user, selecting a specific sample or population and visualizing the results associated by clicking on the 'show target' link.
- Query Genomic Region tab, the tab allows to query the results by using the genomic coordinates. Input a chromosome and start and end value of the region to search, will return a list of targets falling into the region. The results are returned in form of data-table and can be downloaded by exporting the table.
- Graphical Reports tab, the tab contains all the graphics generated by the software, collected into a single place for simplicity and ease of consulting. This tab allows the user to generate any run-time graphic, such as, the population distribution plot and the functional region table, reporting how many targets fell into each functional region and the motif logo of mismatches and bulges, showing the user the base-pairs more prone to have mismatches and/or bulges. The main graphic of the page, is the CFD-score/target count graph, reporting the distribution of the targets ordered by the calculated CFD score, and the differences between reference and alternative targets in the same position, enlightening how the introduction of variants changes the expected outcomes in terms of off-targets.
- Personal Risk Cards tab, the tab contains more specific information about a sample interested by off-targets. This tab, allows the user to perform searches using an individual ID and generates run-time CFD-score/targets count graphics, these two generated graphics, are created by using the 'personal' individual targets, so the targets the individual shares with at least on other individual, and the 'private' targets, so targets that are associated only with the individual of interest. Furthermore, the tab contains a data-table with the 'private' targets and the possibility to download them as a text file.

These are the main components of the user interface developed for CRISPRme, aiming to be simple and understandable by non-informatician. So, the format of pages and the presented data are designed to be friendly for biologists and biotechnologists, as requested by our collaborators.

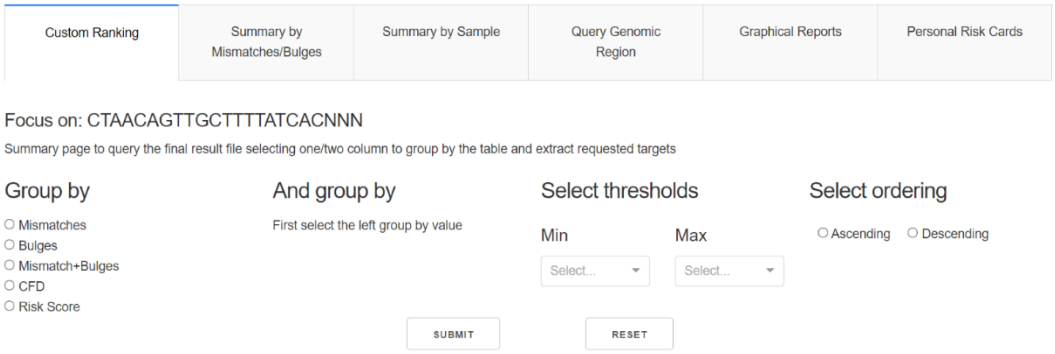


Figure 37. Custom ranking tab, this tab allows the user to perform queries on the targets database with different threshold and group by criteria.

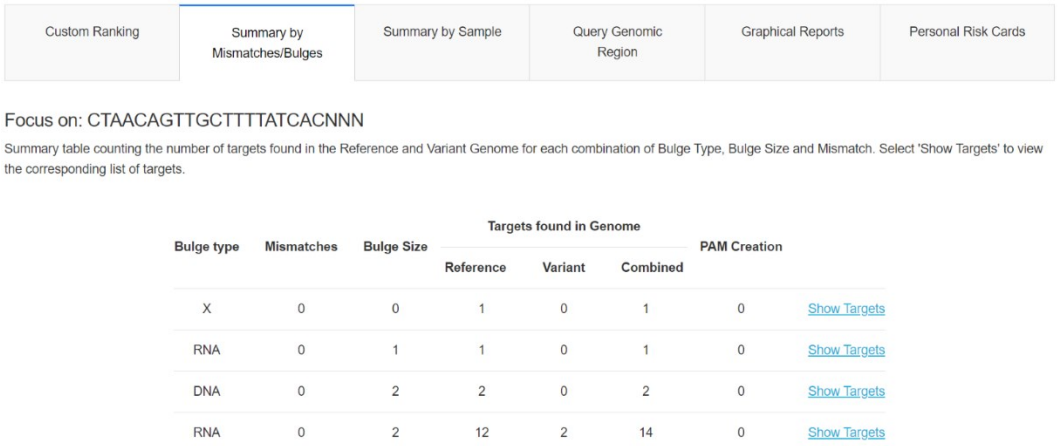


Figure 38. Summary by Mismatches/Bulges tab. This tab separates targets into different categories, by mismatches and bulges and allows the user to visualize the target in each category by using the 'Show Target' link.

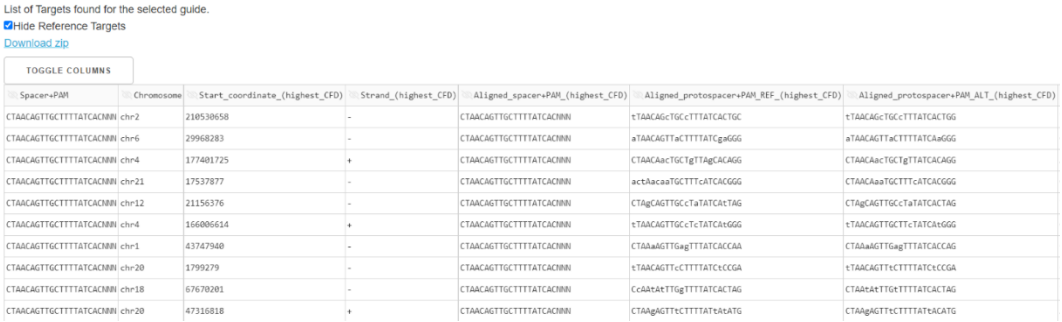


Figure 39. Show Targets table.

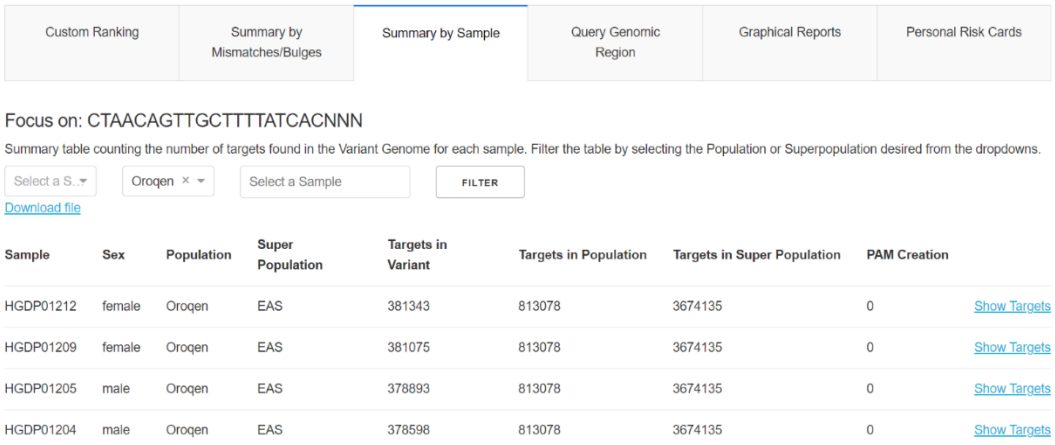


Figure 40. Summary by Sample tab. In this tab the user can search a specific population or sample and extract visualize the related targets, both the counters (Targets in Variant, Targets in Population and Targets in Super Population, PAM creation) and the 'Show Targets' link.

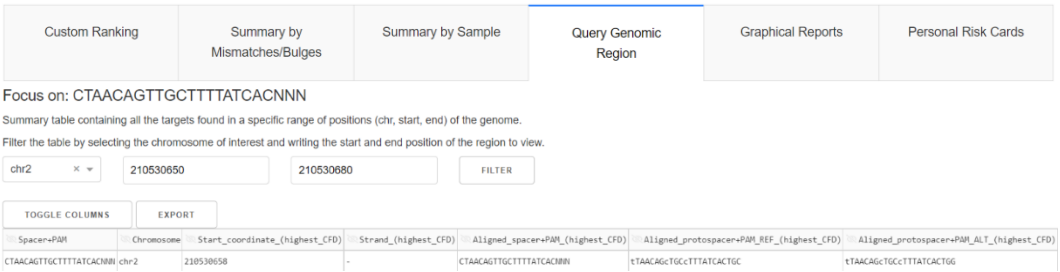


Figure 41. Query Genomic Region tab. In this tab the user can search the results selecting a specific genomic range, obtaining targets in table format.

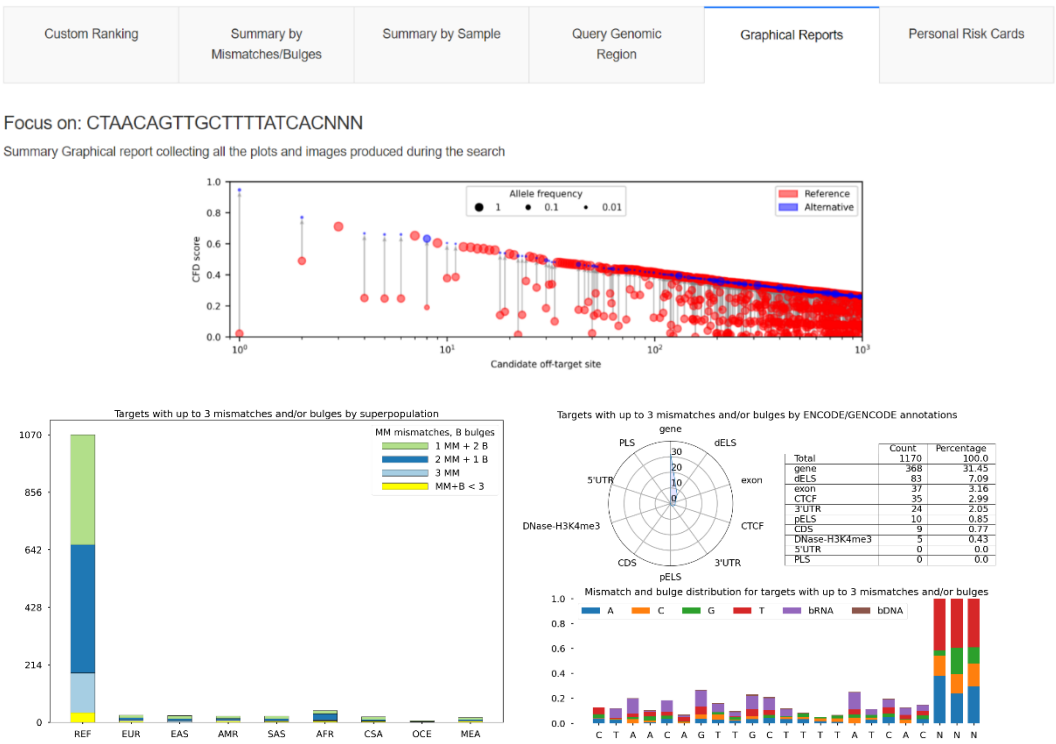


Figure 42. Graphical reports tab. In this tab the user can visualize the cartesian plot with reference and alternative targets ordered by score and connected to represent how much the variant in target, increase the score with respect to reference target. In the tab it's possible to generated bar plots to show the distribution of targets on the populations and how the annotations are distributed on the targets, including a motif logo showing the distribution of mismatches and bulges on the sgRNA sequence.

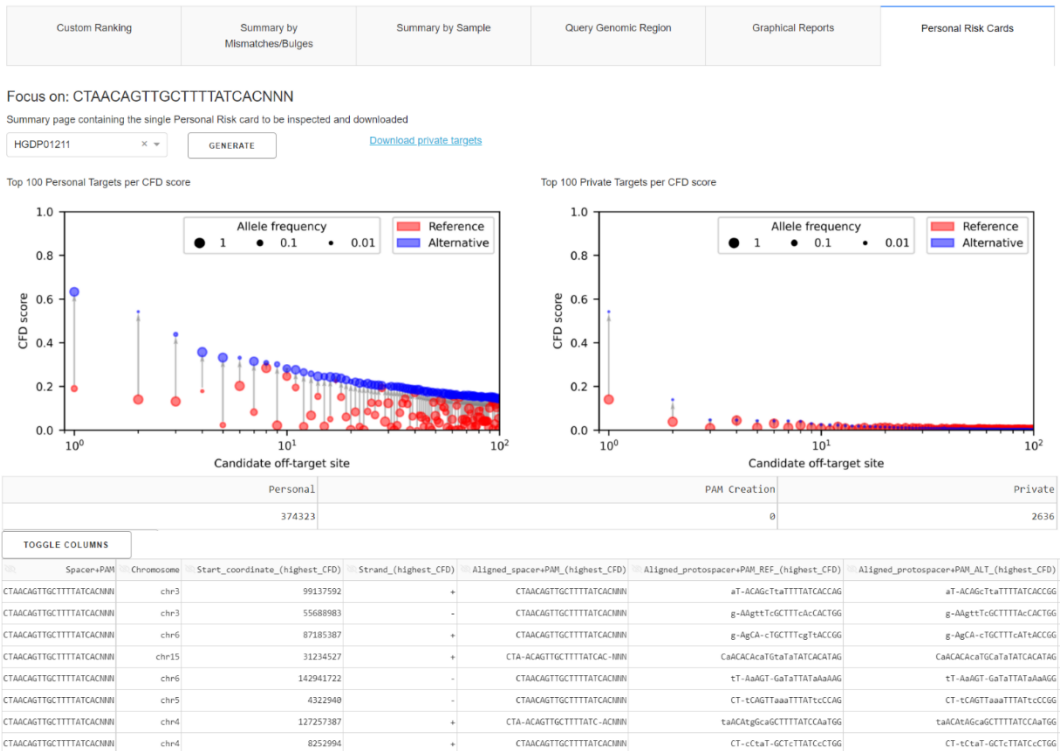


Figure 43. Personal Risk Card tab. In this tab the user can investigate the targets for a specific sample, on the top the presentation of the sample top100 personal targets (left plot) and the top100 private targets (right plot). The table collects all the private targets for the selected sample.

4 Results and Discussion

This section presents the results obtained by CRISPRitz and CRISPRme in terms of targets and off-targets found, produced reports with images and plots, including comparisons with competitors and performance analysis. The section also includes a real case study with results produced by CRISPRme for a therapeutic sgRNA.

Section 4.1 describes the results obtained by CRISPRitz, including off-targets data, profile files with count of off-targets in categories and motif logos of mismatches and bulges distribution on sgRNAs. Furthermore, the section includes comparisons with other state-of-the-art tools, in terms of execution time and found targets.

Section 4.2 describes the results obtained by CRISPRme, starting from CRISPRitz raw targets. Including graphs and plots, variant-induced targets and respective individuals carrying the analyzed targets. CRISPRme also produces many comprehensive files useful to users to generate new pipelines and test panels.

4.1 CRISPRitz results and discussion

This section presents results from CRISPRitz, reporting raw targets file and profile files. It also reports the comparison with other software in terms of execution time and consumed resources.

4.1.1 High-throughput and variant-aware enumeration of potential off-target sites on the functional genome

In some cases, it may be necessary to identify potential off-target sites for a large number of guides, such as for genome-wide CRISPR libraries. In addition, it is often important to evaluate for off-target sites in multiple genomes, such as when performing experiments in multiple cell lines and/or evaluating the effect of personalized variants in individuals (Canver et al., 2018) given that these variants can affect the number of on- and off-target sites or lead to false positive/negative if not accounted for.

Furthermore, once off-target sites are identified, it is useful to inform potential risk in the setting of off-target cleavage through genomic annotation, such as off-target sites in coding sequence or off-targets in non-coding sequence affecting function sequences (e.g., CTCF sites).

Therefore, CRISPRitz was designed to support genetic variants as well as provide intuitive enrichment analysis using a set of genomic annotations for a variety of functional regions, as discussed in Section 2.3 .

To showcase these features, we analyzed the GeCKO Library v2 (Sanjana et al., 2014) genome-wide library, using up to five mismatches (and no bulges). For this analysis we constructed a single enriched genome including 84.4 million genetic variants obtained from 2504 individuals across 26 populations from the 1000 Genome Project Phase 3 (1000 Genomes Project Consortium & others, 2015). CRISPRitz analysis of this dataset with a variant-aware genome and also the reference hg19 genome resulted in a total time of ≃3500 min (2.4 days), for both analyses. Of note, differences were observed between the CRISPRitz results when using the hg19 reference genome as compared with the enriched genome with variants, which is consistent with previous work (we obtain an average increase of ≃50–60% in all the analyzed genetic regions) (Lessard et al., 2017; Scott & Zhang, 2017). It is worth noting, that although this analysis was performed using all the variants present in at least one individual, with CRISPRitz a similar analysis can be performed using variants from a single individual to create a personalized, enriched

genome. The results obtained from this analysis were used as a template for comparison with subsequent analysis of the guides targeting CCR5 (see Figure 45).

Next, we evaluated the guides targeting CCR5 coding sequence, which is a therapeutic target for patients with human immunodeficiency virus (HIV) infection. Using the hg19 and the enriched genome described in the previous section derived from the 1000 Genomes Project database, we observed an increase of 18–24% in putative off-target sites when accounting for SNPs (see Figure 44), which is consistent with previous works. In addition, the visualization of the functional annotation of identified off-target sites can aid in the identification of specific and non-specific guides as shown in Figure 44. We also perform an analysis on the CCR5 guides, including bulges in the search, to show that also the inclusion of bulges is fundamental to obtain an accurate analysis (see Figure 45). The bulges inclusion led to a dramatic increase in the total number of possible off-target sites, from ≃36 000 with four mismatches and no bulges to ≃2.5 million with inclusion of 1 DNA and 1 RNA bulge.

Multiple flavors of high-fidelity Cas9 (e.g. SpCas9-HF1/eSpCas9/HypaCas9/evoCas9) are available that exhibit reduced off-target activity while maintaining on-target editing efficiency (Casini et al., 2018; Chen et al., 2017; Kleinstiver et al., 2016; Slaymaker et al., 2016; Vakulskas et al., 2018). High-fidelity nucleases still maintain probability of cleavage at putative off-target sites albeit a lower probability as compared with standard reagents.

Moreover, use of a high-fidelity Cas9 does not preclude off-target analysis as these nucleases can still mediate off-target cleavage, particularly if a sgRNA is utilized with significant off-target potential.

CRISPRitz analysis may be able to aid nuclease selection prior to initiation of wet-lab experiments as high-fidelity nucleases may be preferred for sgRNAs with increased off-target potential.

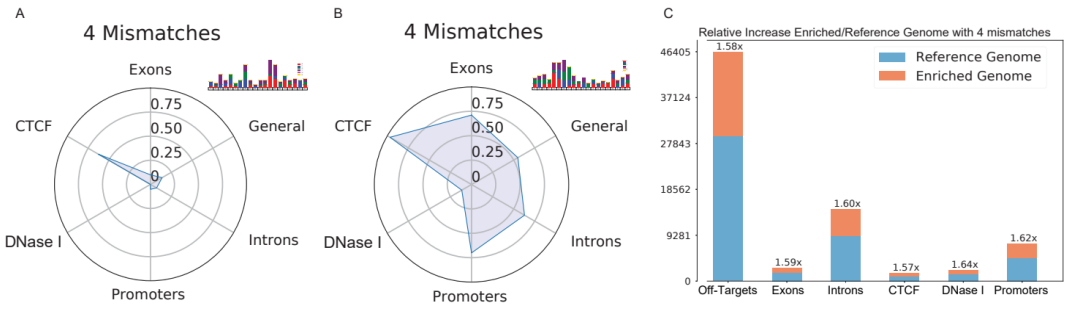


Figure 44. Graphic report from CRISPRitz with two radar charts to understand the distribution of annotations between targets and a bar plot with the distribution of targets on reference and enriched genome.

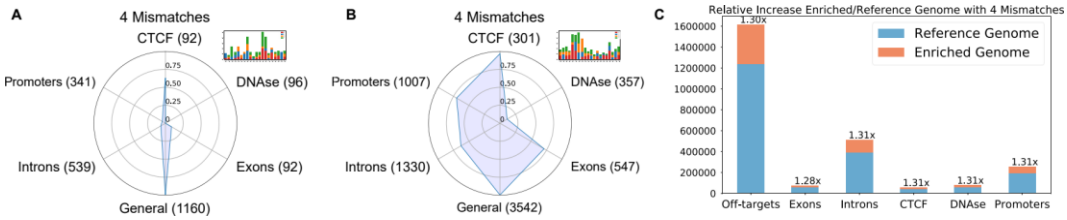


Figure 45. Graphic reports of CRISPRitz using bulges on CCR5 guide.

4.1.2 Performance evaluation and comparison with similar tools

To evaluate the performance of CRISPRitz, we used a general dataset of guides randomly sampled from the human reference genome hg19 using the NGG PAM (see Section 3.1.1). The tests were performed on a machine equipped with an Intel(R) Xeon(R) CPU E5-2650 v4, clocked at 2200 MHz and 64 GBs RAM, and the Ubuntu operating system (version 16.04).

We performed a head-to-head comparison of CRISPRitz with Cas-OFFinder (Bae et al., 2014), FlashFry (McKenna & Shendure, 2018) and OFF-Spotter (Pliatsika & Rigoutsos, 2015) as shown in Figure 46. In Figure 46Figure 46C and Figure 46D, we compared CRISPRitz only with Cas-OFFinder since it was the only available tool (at the time of writing this article), that allowed searches with both mismatches and DNA/RNA bulges. Both CRISPRitz and Cas-OFFinder can take advantage of multi-core architectures. CRISPRitz was implemented using the well-known OpenMP (Dagum & Menon, 1998), while Cas-OFFinder is implemented using OpenCL API (Munshi, 2009b).

First, performance testing without DNA/RNA bulges was performed using a different number of guides with up to five mismatches. FlashFry and OFF-Spotter are faster than CRISPRitz and Cas-OFFinder, ranging from a speed-up of 30–70× comparing CRISPRitz and Flashfry and a ranging from 7 to 25× when comparing CRISPRitz to OFF-Spotter. By using CRISPRitz, we observed an ∼2-fold or greater reduction in execution time as compared with Cas-OFFinder (see Figure 46A). Notably, the execution time slightly increase for CRISPRitz and Cas-OFFinder tools with respect to the number of input guides; however, we observed a significant difference between Cas-OFFinder and CRISPRitz with 1000 guides (Cas-OFFinder performed the search in ≃10 000 s with respect to the ≃3400 s used by CRISPRitz).

Next, using the same hardware, we tested the performance of CRISPRitz and Cas-OFFinder when DNA/RNA bulges were also considered in the analysis. The same guides were tested as in Figure 46Figure 46A and Figure 46B with up to five mismatches but allowing also one DNA and one RNA bulge. Although the execution times are similar for one guide, we observed a 4-fold reduction in execution time for CRISPRitz with respect to Cas-OFFinder when the number of guides increased (Figure 46Figure 46C). Furthermore, the effect of the number of mismatches was evaluated with a fixed number of guides (n = 1000) with the number of DNA and RNA bulges set to one. The difference in execution times was greatest (up to 74-fold reduction) when considering three mismatches (Figure 46Figure 46D). However, the magnitude of execution time reduction decreased to ∼4-fold with an increased number of mismatches. This is because the number of visited branches of the tree (and the execution time consequently) by CRISPRitz increases due to increasing the mismatch and bulge thresholds.

Importantly, CRISPRitz showed robust scalability by varying the number of CPU cores (from 2 to 8), the mismatch threshold from 3 to 5 and the predicted off-target activity per guide (see Figure 47Figure 47). The results highlight that CRISPRitz performance scales approximately linearly over the number of CPU cores (≃1.8).

Finally, we compared the features and the running time of CRISPRitz with additional four software; CRISPOR (Haeussler et al., 2016), CHOPCHOP (Montague et al., 2014), CRISPRseek (Zhu et al., 2014) and CRISPRtool (Lessard et al., 2017), on searching with mismatches and bulges on the reference genome and genome with variants (see Table 3, Table 4, Table 5, Table 6).

CRISPRitz is the only software able to perform a search taking into account genetic variants as well as mismatches and bulges while still maintaining computational efficiency (see Table 7). Furthermore, in a comparison performed with only mismatches allowed in the search, the fastest tool was FlashFry, followed by Off-Spotter, CHOPCHOP and CRISPRitz. Of note, the speed-up range between FlashFry and OFF-

Spotter was from 2- to 4-fold (≃100 s by FlashFry as compared with >400 s by Off-Spotter; see Table 3). In a comparison performed with mismatches allowed in the search and using a variant genome (only supported by CRISPRitz and CRISPRtool), CRISPRitz was the faster tool with a speed-up range of 8.5- to 35-fold (≃3000 s by CRISPRitz as compared >100 000 s taken by CRISPRtool) (see Table 4). In a comparison performed using mismatches and bulges allowed in the search (only supported by CRISPRitz and Cas-OFFinder), we determined that CRISPRitz was the faster tool with a speed-up range from 4- to 75-fold (≃50 000 s by CRISPRitz as compared with >200 000 seconds by Cas-OFFinder; see Table 5). In a comparison performed using mismatches, bulges and a variant genome in the search, only CRISPRitz supported this combination of features with no available alternatives at the time of this article (see Table 6). Taken together, when using only mismatches FlashFry is the fastest tool; however, when bulges or genetic variants are included CRISPRitz offers a significant speedup over all the available tools. Importantly, CRISPRitz is the only tool that allows a complete enumeration of target and off-target sites when accounting simultaneously for mismatches, bulges and genetic variants.

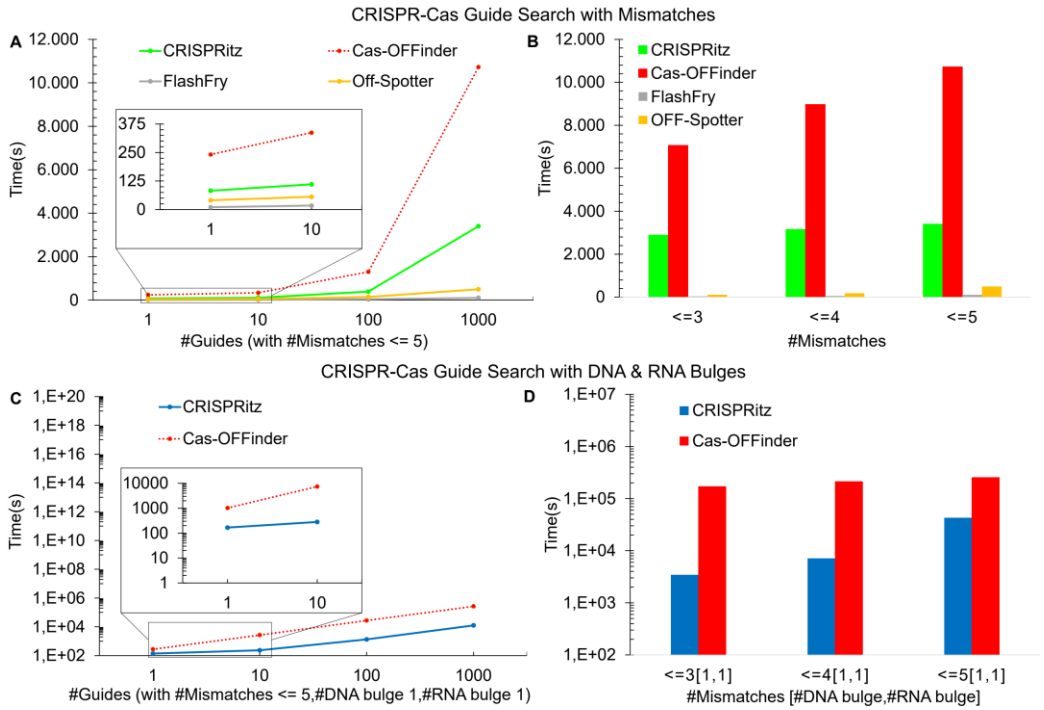


Figure 46. Performance comparison between CRISPRitz, Cas-OFFinder, FlashFry and Off-Spotter. Panel A shows a comparison with increasing set of input guides with <=5 mismatches. Panel B shows a bar plot representation of Panel A. Panel C shows a comparison between CRISPRitz and Cas-OFFinder with <=5 Mismatches and 1 DNA/RNA bulge.

Figure 47Figure 47 shows the scalability of CRISPRitz compared to Cas-OFFinder by varying the number of CPU cores (from 2 to 8) used for the guide search (mismatch threshold of 3, 4, and 5). The figure shows that both the tools scale very well over the number of CPU cores used for the computation. By doubling the number of cores, the obtained speedup is almost linear (1.8x). This underlines the portability of CRISPRitz to parallel architectures. The tests were performed with the random dataset containing 1000 guides, on a machine equipped with an Intel(R) Xeon(R) CPU E5-2650 v4 with 8 cores, clocked at 2200 MHz and 64GBs RAM, and the Ubuntu operating system (version

16.04). Table 2 reports execution time of CRISPRitz and Cas-OFFinder as a function of the number of discovered off-targets per guide.

We sampled 3 groups of 10 guides (based on the hg19 reference genome) with similar but increasing number of off-targets (50, 10000 and 1000000). On this dataset, we run CRISPRitz andCas-OFFinder with 2 cores, 4 mismatches and no bulges. This analysis shows that the number of the off targets per guide has almost no impact on the execution time. Notably, the speed-up of CRISPRitz over Cas-OFFinder is always close to 2.5x.

Software	50 Off-Targets	10000 Off-Targets	1000000 Off-Targets
CRISPRitz	71.82	81.86	82.93
Cas-OFFinder	200.80	201.38	211.53

Table 2. Comparison between CRISPRitz and Cas-OFFinder using guides with different putative off-targets counts to calculated scalability.

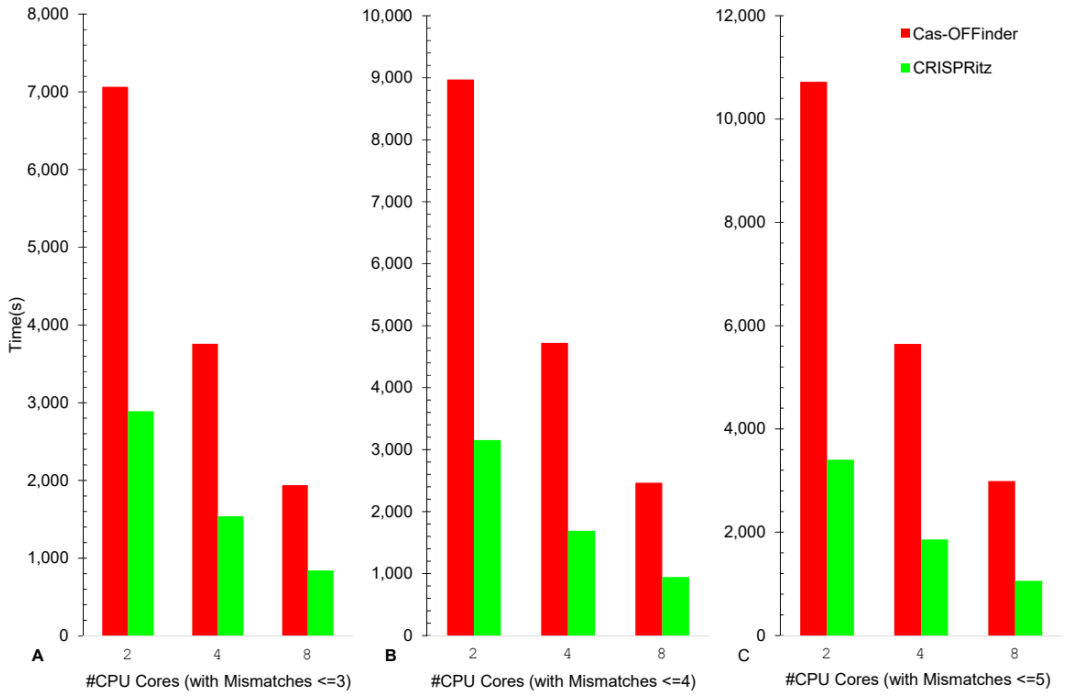


Figure 47. Scalability of CRISPRitz and Cas-OFFinder with increasing number of CPUs cores.

We compared CRISPRitz with other 7 software (Cas-OFFinder (Bae et al., 2014), Flashfry (McKenna & Shendure, 2018), Off-spotter (Pliatsika & Rigoutsos, 2015), CRISPOR (Haeussler et al., 2016), CHOPCHOP (Montague et al., 2014)), CRISPRseek (Zhu et al., 2014)) and CRISPRtool (Lessard et al., 2017) which perform off-target analysis and guide design.

The tests were executed on a machine equipped with an Intel(R) Xeon(R) CPU E5-2650v4, clocked at 2200 MHz and 64 GBs RAM, and Ubuntu operating system (version 16.04).

Experiments were run with 2 cores on CRISPRitz and Cas-OFFinder because they support the parallel execution, and 1 core on all the others, to better reflect a typical mid-size personal computer.

We used a random generated dataset of 1000 guides. In Table 3 the guide search is performed with 3, 4, 5 mismatches and no bulges on the hg19 reference.

As expected, the software that run a pre-processing step to build a database containing candidate targets outperform all the other software if more than one search is done after the database creation. Among the tools that support only searches with mismatches and are based on a precomputed index, Flashfry is the fastest followed by Off-Spotter and CHOPCHOP.

However, among the software that can perform an online search without a pre-processing step, CRISPRitz is the fastest. Table 4 reports the search running time allowing mismatches on the hg19 variant genome with variants from the 1000 Genome project encoded using the IUPAC notation.

CRISPRitz and CRISPRtool are the only 2 software capable of accounting for genetic variants during the search. CRISPRitz outperforms CRISPRtool in every test reaching a speed-up of 35x when using 5 mismatches. In Table 5 we show the results searching with 3, 4, 5 mismatches, 1 DNA and 1 RNA bulges using the hg19 reference.

CRISPRitz and Cas-OFFinder are the only 2 software that can perform this search since the other tools don't support the incorporation of bulges.

In this case, CRISPRitz indexes the reference genome to perform analysis with bulges (see Section 3.1.3). This step is independent of the guide to search and is performed only one time. CRISPRitz outperforms Cas-OFFinder in every test reaching 74x of

speed-up in the 3 mismatches case. Notably, the time required by CRISPRitz

to build the database and run 1 search is about 40x less of the time than CasOFFinder takes to perform 1 search. In Table 6, the guide search is run with

3, 4, 5 mismatches, 1 DNA and 1 RNA bulges and using the hg19 variant genome as defined before.

The only software capable of performing this analysis is CRISPRitz.

Finally, Table 7 summarizes the features, limitations, and capabilities of the existing software. Several software shares the same capabilities and limitations.

Only CRISPRitz and Cas-OFFinder perform bulge analysis.

Only CRISPRitz and CRISPRtool can search using reference genomes and accounting for genetic variants.

Software	3 MM	4 MM	5 MM	DB Time	RAM
CRISPRitz	2890,51	3154,05	3402,73	–	4,26
CAS-OFFinder	7063,56	8969,93	10721,75	–	0,77
FlashFry	40,49	58,44	106,95	3268,29	1,43
OFF-Spotter	114,34	182,28	499,64	743,41	38,09
CRISPOR	5133,99	6160,78	7392,93	3742,59	64,71
CHOPCHOP	1791,01	ND	ND	6744,57	57,41
CRISPRseek	97283,42	100135,78	104152,26	–	65,28
CRISPRtool	20599,78	49637,65	109127,31	–	10

Table 3. Search with 3, 4 and 5 mismatches on reference genome (hg19). Searches were run on a machine with 2 cores for CRISPRitz and Cas-OFFinder, 1 core for all the others. Times are expressed in seconds (3MM, 4MM, 5MM, DB Time) and RAM in GBs.

Software	**3 MM**	**4 MM**	**5 MM**	DB Time	**RAM**
CRISPRitz	2835,65	3078,14	3300,39	–	5,06
CAS-OFFinder	–	–	–	–	–
FlashFry	–	–	–	–	–
OFF-Spotter	–	–	–	–	–
CRISPOR	–	–	–	–	–
CHOPCHOP	–	–	–	–	–
CRISPRseek	–	–	–	–	–
CRISPRtool	24534,67	56345,76	116437,56	–	11,23

Table 4. Search with 3, 4 and 5 mismatches on reference genome (hg19). Searches were run on a machine with 2 cores for CRISPRitz and 1 core for CRISPRtool. Times are expressed in seconds (3MM, 4MM, 5MM, DB Time) and RAM in GBs.

Software	3 MM	4 MM	5 MM	DB Time	RAM
CRISPRitz	2132,54	14258,46	52687,21	1832,24	7,12
CAS-OFFinder	158853,74	201561,34	245363,59	–	2,17
FlashFry	–	–	–	–	–
OFF-Spotter	–	–	–	–	–
CRISPOR	–	–	–	–	–
CHOPCHOP	–	–	–	–	–
CRISPRseek	–	–	–	–	–
CRISPRtool	–	–	–	–	–

Table 5. Search with 3, 4 and 5 mismatches and one bulge (1 DNA/RNA bulge) on reference genome (hg19). Searches were run on a machine with 2 cores for CRISPRitz and Cas-OFFinder. Times are expressed in seconds (3MM, 4MM, 5MM, DB Time) and RAM in GBs.

Software	**3 MM**	**4 MM**	**5MM**	DB Time	**RAM**
CRISPRitz	8627,13	57475,97	210748,84	1832,24	8,17
CAS-OFFinder	–	–	–	–	–
FlashFry	–	–	–	–	–
OFF-Spotter	–	–	–	–	–
CRISPOR	–	–	–	–	–
CHOPCHOP	–	–	–	–	–
CRISPRseek	–	–	–	–	–
CRISPRtool	–	–	–	–	–

Table 6. Search with 3, 4 and 5 mismatches and one bulge (1 DNA/RNA bulge) on variant genome (hg19 with 1000G). Search was run on a machine with 2 cores for CRISPRitz. Times are expressed in seconds (3MM, 4MM, 5MM, DB Time) and RAM in GBs.

Software	MM	Bulges	Enzymes	Genomes	IUPAC	Score	Annotation
CRISPRitz	0-Any	0-Any	User specified	Every fasta genome in UCSC format	Yes	Doench2016azimuth, CFD	Any
CAS-OFFinder	0-Any	0-Any	User specified	Every fasta genome in UCSC format	No	None	None
FlashFry	0-Any	None	SpCas9, Cpf1	Every fasta genome in UCSC format	No	Hsu2013, Doench2014, Doench2016azimuth, Moreno-Mateos	Any
OFF-Spotter	0-5	None	SpCas9NGG, SpCas9NAG, saCas9, CjCas9	GRCh38, GRCh37, GRCm38, w303	No	User customizable	Any
CRISPOR	0-5	None	SpCas9, saCas9, CjCas9, Cpf1	Every fasta genome in UCSC format	No	MIT, CFD, Doench2016azimuth, Doench2016old, Chari, Xu, Wu-Crispr, Doench2014, Wang, Moreno-Mateos, Azimuth in-Vitro, CCTop, deepCpf1, Najm et al 2018	Any
CHOPCHOP	0-3	None	User specified	very fasta genome in UCSC format	No	Doench2016azimuth, Chari, Xu, Doench2014, Moreno-Mateos	Any
CRISPRseek	0-Any	None	User specified	All BSgenome available	No	Hsu 2013	mRNA exons
CRISPRtool	0-Any	None	User specified	All BSgenome available	Yes	Sanjan2014, Hsu2013, CFD	None

Table 7. Comparison of features between CRISPRitz and other software.

4.1.3 Discussion

CRISPRitz was designed and presented as an all-in-one solution, with the aim of helping technician without a computational and programming background, performing a complex task as CRISPR/Cas off-targets prediction and enumeration.

The software was developed with this aim in mind and encloses this objective in the entirety of the project.

CRISPRitz can be used to perform off-target enumeration and prediction, to score targets using CFD and Doench scores, to retrieve functional and genetical annotations, to find variant-induced targets and finally plotting all these data in graphical reports and summarized matrix file.

So, CRISPRitz can be used by non-expert to obtain simple and exhaustive results, without the necessity of complex and manual operations, with the possibility of using many different types of nucleases, sgRNAs and genomes. Including also variants and BED files with annotations.

CRISPRitz is an all-in-one solution and toolbox for any technician and biotechnologist that need to perform in-silico CRISPR/Cas analysis and predictions. We believe also the comparison presented in Section 4.1.2 highlights the utility and flexibility of CRISPRitz in performing exhaustive and complete searches using variant genomes, any number of mismatches and bulges, support for arbitrary PAM, activity and off-target scores and providing visual reports based on user defined functional annotations.

4.2 CRISPRme results and discussion

This section presents a real case study processed with CRISPRme, reporting the results, the graphs and a new discovered target that led to the development of a new project. The section also includes computational performances comparisons with other software and tools.

4.2.1 CRISPRme functionalities and operational process

CRISPRme is a web-based tool to predict off-target potential of CRISPR gene editing that accounts for genetic variation. CRISPRme takes as input a Cas protein, gRNA spacer sequence(s) and PAM, genome build, sets of variants (VCF files for populations or individuals), user-defined thresholds of mismatches and bulges, and optional user-defined genomic annotations to produce comprehensive and personalized reports (see Figure 48).

We have designed CRISPRme to be flexible with support for new gene editors with variable and extremely relaxed PAM requirements (Walton et al., 2020). Thanks to a PAM encoding based on Aho-Corasick automata (see Section 3.1.1) and an index based on a ternary search tree (see Section 3.1.3), CRISPRme can perform genome-wide exhaustive searches efficiently even with an NNN PAM, extensive mismatches (tested with up to 7) and RNA:DNA bulges (tested with up to 2).

Notably, a comprehensive search performed with up to 6 mismatches, 2 DNA/RNA bulges and a fully non-restrictive PAM (NNN) takes only 19 hours on a small computational cluster (Intel Xeon CPU E5-2609 v4 clocked at 2.2 GHz and 128 GB RAM). All the 1000G variants, including both SNVs and indels, can be included in the search together with all the available metadata for each individual (sex, super-population and age), and the search operation takes into account observed haplotypes (see Section 4.2.2). Importantly, off-target sites that represent alternative alignments to a given genomic region are merged to avoid inflating the number of reported sites. Although several tools exist to enumerate off-targets, to our knowledge only a command line tool called crispRtool (Lessard et al., 2017) incorporates genetic variants in the search. However, it has a search operation limited to 5 mismatches, cannot include DNA or RNA bulges, does not provide a graphical interface and is orders of magnitude slower than CRISPRme (see Section 4.2.4).

CRISPRme generates several reports (see Section 4.2.3). First, it summarizes for each gRNA all the potential off-targets found in the reference or variant genomes based on their mismatches and bulges (see Figure 32) and generates a file with detailed information on each of these candidate off-targets. Second, it compares gRNAs to customizable annotations. By default, it classifies possible off-target sites based on GENCODE (Frankish et al., 2019) (genomic features) and ENCODE (E. P. Consortium & others, 2004) (candidate cis-regulatory elements, cCREs) annotations. It can also incorporate user-defined annotations in BED format, such as empiric off-target scores or cell type specific chromatin features. Third, using 1000G (1000 Genomes Project Consortium & others, 2015) and/or HGDP (Foster, 2008) variants, CRISPRme reports the cumulative distribution of homologous sites based on the reference genome or super-population. These global reports could be used to compare a set of gRNAs to demonstrate the frequency of allele-specific off-target sites across individuals and how genetic variation impacts the predicted cleavage potential using cutting frequency determination (CFD) scores (Doench et al., 2016b). Finally, CRISPRme can generate personal genome focused reports called personal risk cards (see Figure 43). These reports indicate off-target sites modified by private genetic variants not found in reference genome.

Figure 48. CRISPRme process. Panel a) presents the main input page. Panel b) contains the outcomes from a real case study. Panel c) enlightens one variant off-target generated by a specific SNP in PAM sequence. Panel d) present the cartesian plot with reference and alternative targets ordered by CFD score showing the increase in score value due to variant introduction. Panel e) shows the position of the candidate off-target with SNP.

4.2.2 Real Case study and outcomes

We tested CRISPRme with a gRNA (#1617) targeting a GATA1 binding motif at the +58 erythroid enhancer of BCL11A (Canver et al., 2015; Karczewski et al., 2020). A recent clinical report described two patients, one with SCD and one with β-thalassemia, each treated with autologous gene modified hematopoietic stem and progenitor cells (HSPCs) edited with Cas9 and this gRNA, who showed sustained increases in fetal hemoglobin, transfusion-independence and absence of vaso-occlusive episodes (in the SCD patient) following therapy (Stadtmauer et al., 2020)(. This study as well as prior pre-clinical studies with the same gRNA (#1617) did not reveal evidence of off-target editing in treated cells considering off-target sites nominated by bioinformatic analysis of the human reference genome and empiric analysis of in vitro genomic cleavage potential (Canver et al., 2015; Moore et al., 2020; Stadtmauer et al., 2020). CRISPRme analysis found that the predicted off-target site with both the greatest CFD score and the greatest increase in CFD score from the reference to alternative allele was at an intronic sequence of CPS1 (see Figure 48), a genomic target subject to common genetic variation (modified by a SNP with MAF ≥ 1%). CFD scores range from 0 to 1, where the on-target site has a score of 1. The alternative allele rs114518452-C generates a TGG PAM sequence (that is, the optimal PAM for SpCas9) for a potential off-target site with 3 mismatches and a CFD score (CFDalt 0.95) approaching that of the on-target site (see Figure 48Figure 48). In contrast, the reference allele rs114518452-G disrupts the PAM to TGC, which markedly reduces predicted cleavage potential (CFDref 0.02). rs114518452-C has an overall MAF of 1.33% in gnomAD (Karczewski et al., 2020) v3.117, with MAF of 4.55% in African/African-American, 0.02% in European (non-Finnish) and 0.00% in East Asian super-populations (see Figure 48Figure 48).

To consider the off-target potential that could be introduced by personal genetic variation that would not be predicted by 1000G variants, we analyzed HGDP variants identified from whole genome sequences of 929 individuals from 54 diverse human populations (Lowy-Gallego et al., 2019). We observed 249 candidate off-targets with CFD ≥0.2 for which the CFD score in HGDP exceeded that found for either the reference genome or 1000G variants by at least 0.1 (see Figure 49Figure 49). These additional variant off-targets not found from 1000G were observed in each super-population, with the greatest frequency in the African super-population (see Figure 49). 229 (92.0%) of these variant off-targets not found in 1000G were unique to a super-population and 172 (69.1%) of these were unique to just one individual (see Figure 49Figure 49). Single individual focused searches, for example an analysis of HGDP01211, an individual of the Oroqen population within the East Asian super-population, showed that most variant off-targets (with higher CFD score than reference) were due to variants also found in 1000G (n=32369, 90.4%), a subset were due to variants shared with other individuals from HGDP but absent from 1000G (n=3177, 8.9%), and a small fraction were private to the individual (n=234, 0.7%) (see Figure 49). Among these private off-targets was one generated by a variant (rs1191022522, 3-99137613-A-G, gnomAD v3.1 MAF 0.0053%) where the alternative allele produces an NGG PAM that increases the CFD score from 0.14 to 0.54 (see Figure 49).

To experimentally test the top predicted off-target from CRISPRme, we identified a CD34+ HSPC donor of African ancestry heterozygous for rs114518452-C (the variant predicted to introduce the greatest increase in off-target cleavage potential; see Figure 48). We performed RNP electroporation using a gene editing protocol that preserves engrafting HSC function19. Amplicon sequencing analysis showed 92.0 ± 0.5% indels at the on-target site and 4.8 ± 0.5% indels at the off-target site. Evaluable indels were strictly found at the alternative PAM-creation allele without indels observed at the reference allele (see Figure 51Figure 51), suggesting 9.6 ± 1.0% off-target editing of the

alternative allele. In an additional 6 HSPC donors homozygous for the reference allele rs114518452-G/G, 0.00 ± 0.00% indels were observed at the off-target site, suggesting strict restriction of off-target editing to the alternative allele (see Figure 49Figure 49).

These results demonstrate how personal genetic variation may influence the off-target potential of therapeutic gene editing. In the case of BCL11A enhancer editing, up to ~10% of SCD patients with African ancestry would be expected to carry at least one rs114518452-C allele. In general, therapeutic gene editing clinical trials might consider evaluating the impact of population and private genetic variation on gene editing outcomes including individual patient assessment.

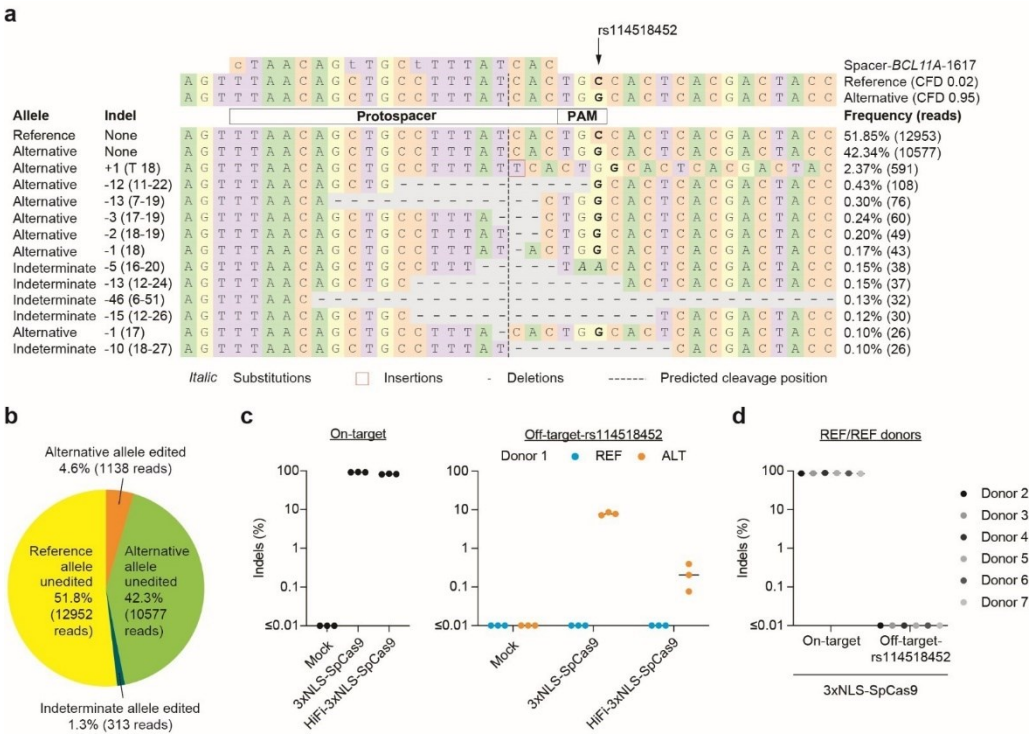


Figure 49. Allele-specific off-target editing by a BCL11A enhancer targeting gRNA associated with a common variant in African-ancestry populations. Panel a) Human CD34+ HSPCs from a donor heterozygous for rs114518452-G/C (Donor 1, REF/ALT) were subject to 3xNLS-SpCas9:sg1617 RNP electroporation followed by amplicon sequencing of the off-target site around chr2:210,530,659-210,530,681 (off-target-rs114518452 in 1-start hg38 coordinates). CFD scores for the reference and alternative alleles are indicated and representative aligned reads are shown. Spacer shown as DNA sequence for ease of visual alignment, with mismatches indicated by lowercase and the rs114518452 position shown in bold. Coordinates are for hg38 and 1-start. Panel b) Reads classified based on allele (indeterminate if the rs114518452 position is deleted) and presence or absence of indels (edits). Panel c) Human CD34+ HSPCs from a donor heterozygous for rs114518452-G/C (Donor 1) were subject to 3xNLS-SpCas9:sg1617 RNP electroporation, HiFi-3xNLS-SpCas9:sg1617 RNP electroporation, or no electroporation (mock) followed by amplicon sequencing of the on-target and off-target-rs114518452 sites. Each dot represents an independent biological replicate (n = 3). Indel frequency was quantified for reads aligning to either the reference or alternative allele. Panel d) Human CD34+ HSPCs from 6 donors homozygous for rs114518452-G/G (Donors 2-7, REF/REF) were subject to 3xNLS-SpCas9:sg1617 RNP electroporation with 1 biological replicate per donor followed by amplicon sequencing of the on-target and OT-rs114518452 sites.

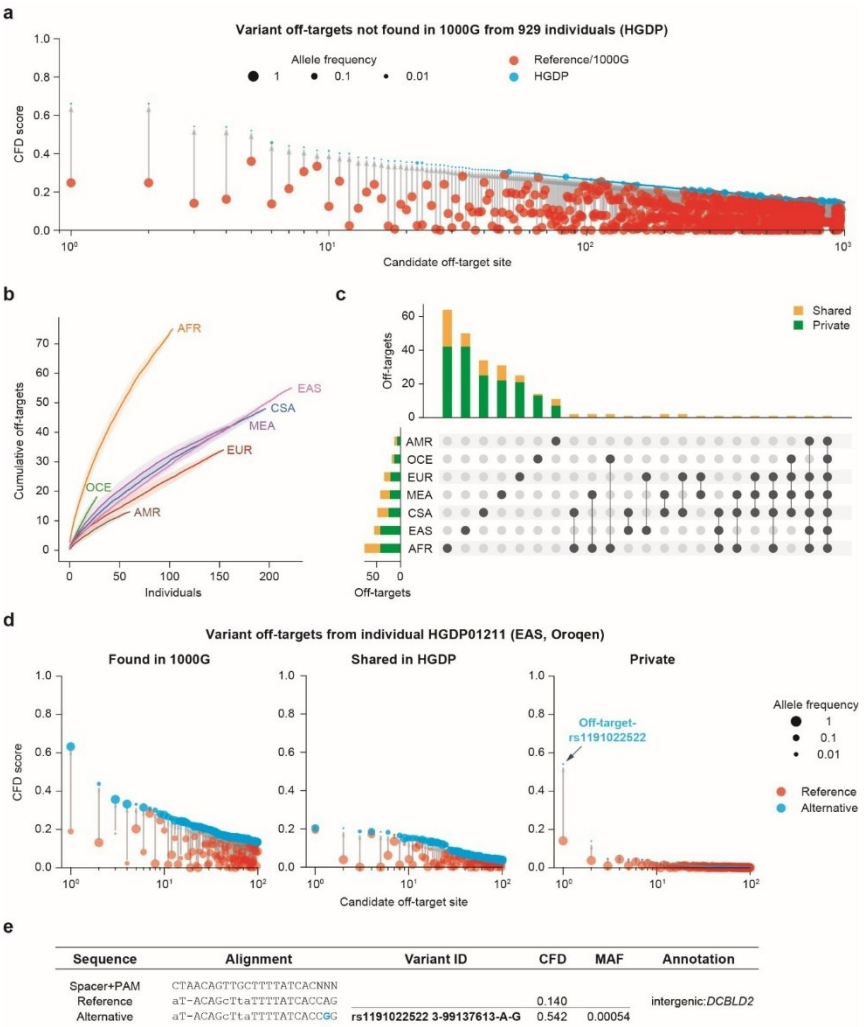


Figure 50. Results produced using Human Genome Diversity Project (HGDP) variant dataset. Panel a) shows cartesian with HGDP-only targets, showing how the inclusion of new variants, introduces many new putative off-targets. Panel b) shows the distribution of off-targets on different populations. Panel c) shows overlapping targets for each population. Panel d) presents an analysis at individual level for sample HGDP01211. shared with 1000G variant off-targets (left panel), higher CFD score compared to reference genome and 1000G but shared with other HGDP individuals (center panel), and higher CFD score compared to reference genome and 1000G with variant not found in other HGDP individuals (right panel). For the center and right panels, reference refers to CFD score from reference genome or 1000G variants. Panel e) enlighten the top predicted target for HGDP01211 with SNP.

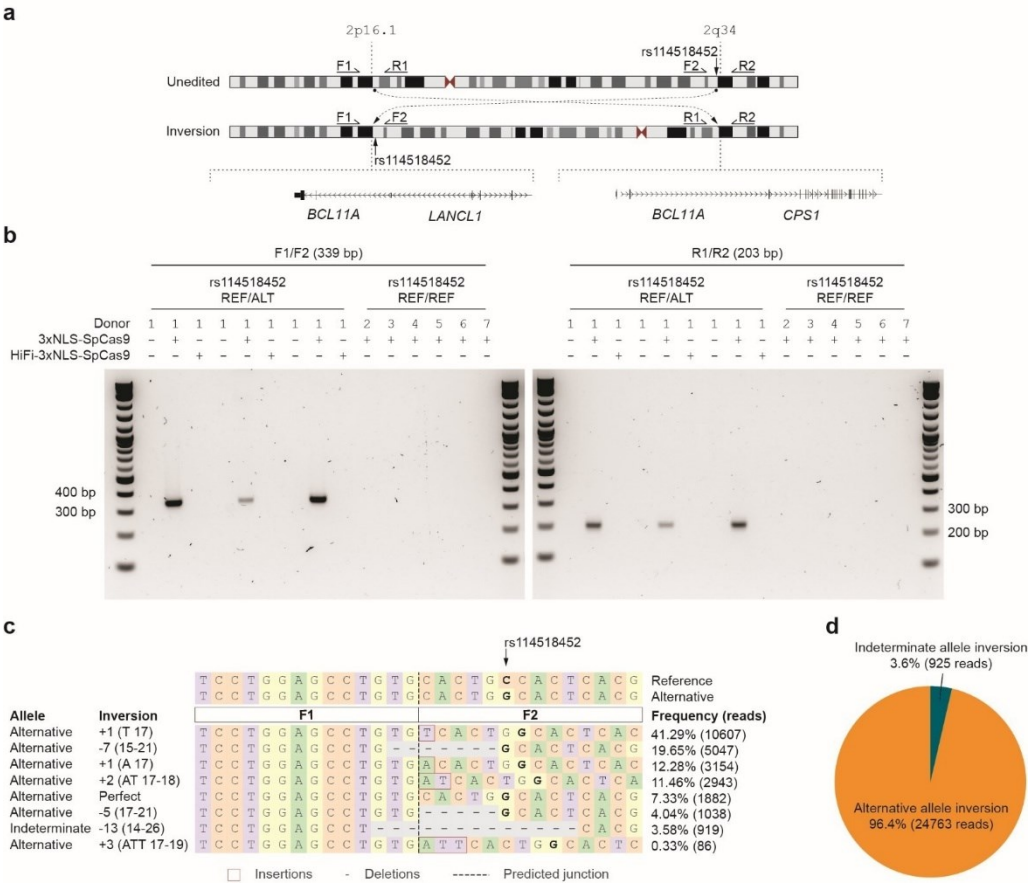


Figure 51. Allele-specific pericentric inversion following BCL11A enhancer editing. Panel a) Concurrent cleavage of the on-target and off-target-rs114518452 sites could lead to pericentric inversion of chr2 as depicted. PCR primers F1, R1, F2, and R2 were designed to detect potential inversions. Panel b) Human CD34+ HSPCs from a donor heterozygous for rs114518452-G/C (Donor 1) were subject to 3xNLS-SpCas9:sg1617 RNP electroporation, HiFi-3xNLS-SpCas9:sg1617 RNP electroporation, or no electroporation with 3 biological replicates. Human CD34+ HSPCs from 6 donors homozygous for rs114518452-G/G (Donors 2-7, REF/REF) were subject to 3xNLS-SpCas9:sg1617 RNP electroporation with 1 biological replicate per donor. Gel electrophoresis for inversion PCR performed with F1/F2 and R1/R2 primer pairs on left and right respectively with expected sizes of precise inversion PCR products indicated. Panel c) Reads from amplicon sequencing of the F1/F2 product (expected to include the rs114518452 position) from 3xNLS-SpCas9:sg1617 RNP treatment were aligned to reference and alternative inversion templates. The rs114518452 position is shown in bold. Panel d) Reads classified based on allele (indeterminate if the rs114518452 position deleted).

4.2.3 Reports and individual graphs

CRISPRme reference/alternative CFD comparison obtained using sg1617, NNN PAM and 6 mismatches plus 2 DNA/RNA bulges, tested on the hg38 reference genome plus 1000G and HGDP variants (see Figure 50Figure 50). A stem plot shows the distribution of CFD scores for candidate off-targets ranked in descending order by CFD score. For candidate off-targets for which a genetic variant increases the CFD score, the CFD scores for both the alternative (blue) and reference (red) allele at the same locus is shown. The area of the circle is proportional to allele frequency. b) On the left, stacked bar plots summarizing the number of candidates off-targets for each category of mismatch + bulge in super-populations present in the input variant data. On the right, radar chart showing the percentage of off-targets falling into a specific genomic annotation with respect to the overall count, table detailing the exact number of off-targets falling into each category (an off-target can fall under more than one category) and motif plot showing the distribution of mismatches and bulges with respect to the spacer+PAM.

CRISPRme provides a dedicated page to generate reports called Personal Risk Cards (see Figure 52Figure 52) that summarize potential off-target editing by a particular gRNA in a given individual due to genetic variants. This feature is particularly useful for retrieving and investigating private off-targets.

The report contains two dynamically generated plots depicting all the candidate variant off-targets for the sample including those non-unique to the individual and those that are unique to the individual. These plots highlight how the introduction of genetic variants can change the predicted off-target cleavage potential, thereby demonstrating the importance of variant-aware off-target assessment as in CRISPRme. The report also contains two tables (see Figure 52Figure 52), consisting of a summary (Table 1, top) and information on each extracted candidate off-target (Table 2, bottom) with the following columns:

Table 1:

Personal, count of all the candidate variant off-targets for the selected sample (including both variants unique and non-unique to the individual).

PAM creation, count of all the instances where a genetic variant in the sample introduces a new PAM and the PAM used in the search in not found in the reference genome at the same locus.

Private, count of all the candidate variant off-targets uniquely found in the selected sample.

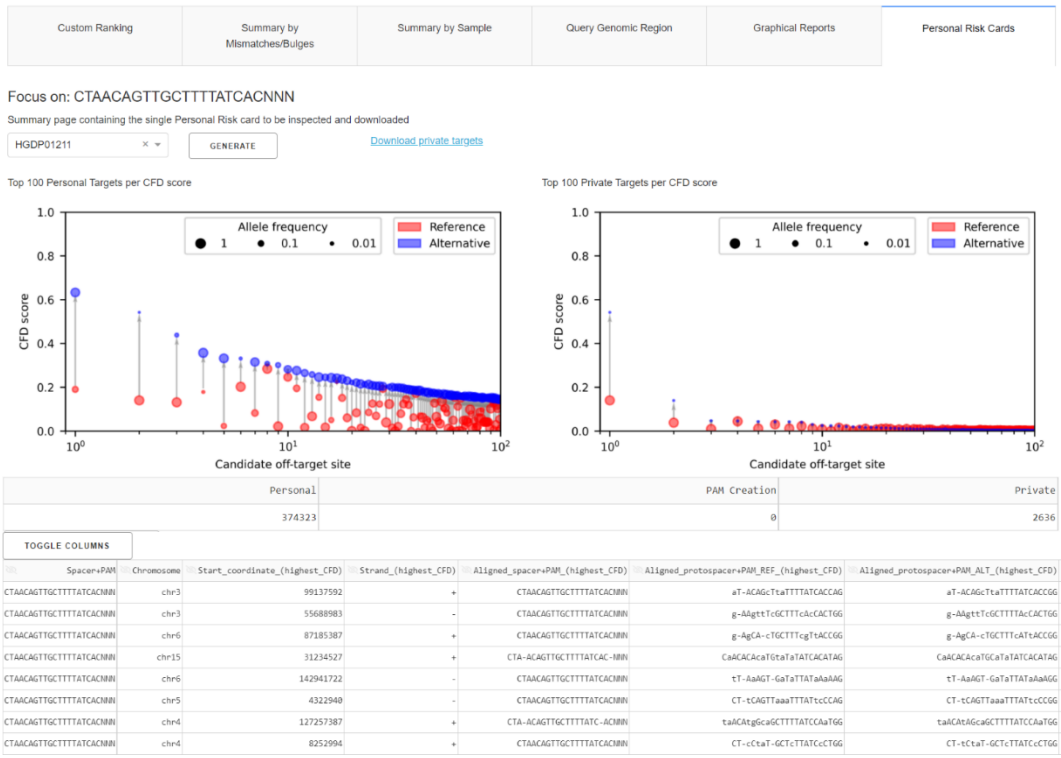


Figure 52. Personal Risk Card.

4.2.4 Comparison with available tools

Although numerous tools are available to enumerate CRISPR-Cas off-targets, to our knowledge only two previous studies (Chaudhari et al., 2020; Lessard et al., 2017) have reported computational strategies to assess off-target potential in presence of genetic variants. Only crispRtool from (Lessard et al., 2017) provides a general command line software. Therefore, we decided to focus our comparison by assessing the features and running times of CRISPRme (v1.7.7) and crisprRtool (v2.0.5) on the same hardware (AMD Ryzen Threadripper 3970X 32-Core Processor clocked at 2.2 GHz with 124 GB RAM) to provide a fair assessment. For our tests we used the 1617 sgRNA, NGG PAM, variants from 1000G and a variable number of mismatches and bulges.

Briefly, crisprRtool first adds variants (SNPs only) to the reference genome using IUPAC notation, and then searches the input gRNA(s) on the variant genome and reports a list of putative on-and off-targets with IUPAC nucleotides. The tool also offers the possibility to search each VCF file individually to resolve haplotypes (SNPs and INDELs) of the reported off-targets. However, for this step, the user needs to manually edit and execute a script for each VCF file. In addition, the search operation with crispRtool allows a maximum of 5 mismatches, does not account for bulges, and is not flexible in terms of PAM location relative to the protospacer (only 3' is supported).

Using 5 mismatches and the settings described above, crispRtool took 9 hours to complete the non-haplotype-resolved search. The haplotype-resolved search only on chr1 using variants from 1000G (6 million SNPs and INDEL variants) took ~37 hours. Conservatively extrapolating to all other chromosomes, the entire search would take more than 300 hours and will not be as complete as the search CRISPRme offers due to the lack of graphical reports and textual summaries encompassing results from all chromosomes.

On the other hand, by leveraging an efficient genome index and auxiliary data structures that are constructed only once during the installation (~4 hours for NGG PAM, ~12 hours for NNN PAM or can be downloaded directly in our complete test package), CRISPRme can complete a haplotype-aware search for a gRNA across the entire genome with 5 mismatches in ~1 hour. The haplotype-resolved search on the entire genome with up to 6 mismatches and 2 DNA/RNA bulges only takes 2 hours (excluding the guide-independent indexing operation described above) and includes a summary report.

4.2.5 Discussion

CRISPRme was designed and developed to help technician and biotechnologist, with an easy-to-use tool to perform analysis and visualize CRISPR/Cas results.

CRISPRme incorporates CRISPRitz powerful search tools and integrates it, with a GUI and more individual centric analysis.

CRISPRme allows the user to perform extensive searches with reference and variant-enriched genomes, including also functional annotations and genes definition. The software takes care of all the necessary steps and returns a comprehensive result directory, will a complete result file, matrix-like files with reports and graphs.

The aim of CRISPRme is to simplify the work of the user by collecting and visualize all possible targets and present them in an easy-to-read setting.

In fact, the software uses a web-based GUI to present the data to users, allowing them to find targets with specific mismatches and bulges count or variant-induced targets related to a specific sample. The software also returns counters with collected results, like a general counter informing the user with the number of targets accounted with a specific count of mismatches and bulges or targets found with a specific type of bulge. The software also presents run-time graphs, generate on request when the user select a combination of mismatches and bulges.

So, the aim of the software is to help user with the understanding of a complex problem supporting the search with a set of files and processed data.

The results are generated in accordance with comments and requests done by our collaborators, technicians and biotechnologists, with the purpose of obtaining significant targets and intelligible results.

In summary, CRISPRme answers to a precise necessity, having an all-in-one toolbox, usable by non-informaticians, that process huge quantity of data, solves a complex task like off-targets enumeration and returns many simple and easy-to-read and use files and plots.

5 Future Directions

This section summarizes new implications and possible directions of the field.

Genetic engineering and all its correlated fields and sectors have seen a huge growth in the last years. Leading to an increase interest in the sector and resulting in an explosion of project and related works.

The work presented in this thesis, is directed to professionals that want to understand and estimate outcomes of experiments without the costs in terms of money, equipment and time. This trend is becoming very popular due to the decreasing of price of powerful computers, in conjunction with the ease of use and simplicity of prediction software, like the ones presented in this work.

So, future directions of the fields will be represented by the use of these software and similar software in prediction pipelines, that will become more and more popular amongst technicians due to necessity and ease of use.

Since CRISPR/Cas is becoming predominant and it is ever-growing, prediction pipelines and genetic screening will become standard for any medical treatment and personal medicine (Flores et al., 2013) will become the standardized approach in the recent future.

Furthermore, useful computational applications and prediction tools, will become the standard approach to avoid wasting resources and time to test with a blind approach.

In a near future, a patient will enter a hospital, got sequenced, tested for any possible harmful predicted off-target or possible susceptibility to a drug or a treatment, and cured with its specific condition as a starting point.

Similar software will become more and more popular, they will be integrated with chemical and biophysical knowledge, correcting the result using the ever-growing knowledge about CRISPR/Cas behavior. Right now, is nearly impossible to predict outcomes without using a brute force approach and then filtering results by using empirical data, regression algorithm and machine learning techniques, trying to extrapolate real off-targets by comparing expected results with obtained results.

This task is now time consuming, imprecise and generates many false positives.

In fact, is not possible to certificate in-silico CRISPR/Cas outcome, without performing in vitro and in vivo sequencing.

What is expected is to resolve these problems, by producing more and more accurate models to predict outcome without the usage of brute force and exhaustive approaches, resulting in fewer computational time, less junk results and more accuracy in the produced results.

Another aspect that will surely become more and more studied in the future, is the usage of variants and mutations.

In this work, it’s explained how to use and why it is necessary to use variants to find possible off-target, including also a real case study showing the importance of this approach.

In the future, it is expected the growth and the production of more and more variants databases, containing more samples, more specific mutations and with more accuracy.

This will dramatically increase the power of prediction tools and will transform them in certifier and assessment tools, transforming the approach from a brute force, omni-comprehensive result producer, to a specific and on-point analysis, resulting in precise and time saving analysis. This will completely transform the field and how the computational approach is seen and used.

Summarizing, future directions for the field, will be:

- Possibility to transform brute-force and exhaustive approach, to a more precise and on-point approach, reducing time wasting and resource consumption.
- Utilization of personal data, as variants and medical records, to direct the searches on-point, finding only the outcomes that are correlated to a specific person or pathology.
- Substitution of time consuming and expensive lab tests, with faster, cheaper and accurate in-silico predictions.

These directions will be taken in less years than expected and will revolutionize how we think and how we approach these kinds of problems.

6 References

A map of human genome variation from population-scale sequencing. (2010). *Nature*, *467*(7319), 1061–1073. https://doi.org/10.1038/nature09534

Aho, A. v, & Corasick, M. J. (1975). Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, *18*(6), 333–340.

Allen, G., & Owens, M. (2010). *The Definitive Guide to SQLite*. Apress. https://doi.org/10.1007/978-1-4302-3226-1

Auer, T. O., Duroure, K., Concordet, J.-P., & del Bene, F. (2014). CRISPR/Cas9-mediated conversion of eGFP- into Gal4-transgenic lines in zebrafish. *Nature Protocols*, *9*(12), 2823–2840. https://doi.org/10.1038/nprot.2014.187

Bae, S., Park, J., & Kim, J.-S. (2014). Cas-OFFinder: a fast and versatile algorithm that searches for potential off-target sites of Cas9 RNA-guided endonucleases. *Bioinformatics*, *30*(10), 1473–1475. https://doi.org/10.1093/bioinformatics/btu048

Bayer, R., & McCreight, E. M. (1972). Organization and maintenance of large ordered indexes. *Acta Informatica*, *1*(3), 173–189. https://doi.org/10.1007/BF00288683

Bemer, R. W. (1961). Letters to the editor. *Communications of the ACM*, *4*(3). https://doi.org/10.1145/366199.366206

Bentley, J., & Sedgewick, B. (1998a). Ternary search trees. *Dr. Dobb's Journal*, *23*(4).

Bentley, J., & Sedgewick, B. (1998b). Ternary search trees. *Dr. Dobb's Journal*, *23*(4).

Boyer, R. S., & Moore, J. S. (1977). A fast string searching algorithm. *Communications of the ACM*, *20*(10), 762–772. https://doi.org/10.1145/359842.359859

Bron, C. (1972). Merge sort algorithm [M1]. *Communications of the ACM*, *15*(5), 357–358. https://doi.org/10.1145/355602.361317

Browser, U. G. (2013). Human Genome 19. *Http://Hgdownload.Cse.Ucsc.Edu/GoldenPath/Hg19/Chromosomes/*.

Cancellieri, S., Canver, M. C., Bombieri, N., Giugno, R., & Pinello, L. (2020). CRISPRitz: rapid, high-throughput and variant-aware in silico off-target site identification for CRISPR genome editing. *Bioinformatics*, *36*(7), 2001–2008. https://doi.org/10.1093/bioinformatics/btz867

Canver, M. C., Joung, J. K., & Pinello, L. (2018). Impact of genetic variation on CRISPR-Cas targeting. *The CRISPR Journal*, *1*(2), 159–170.

Canver, M. C., Smith, E. C., Sher, F., Pinello, L., Sanjana, N. E., Shalem, O., Chen, D. D., Schupp, P. G., Vinjamur, D. S., Garcia, S. P., Luc, S., Kurita, R., Nakamura, Y., Fujiwara, Y., Maeda, T., Yuan, G.-C., Zhang, F., Orkin, S. H., & Bauer, D. E. (2015). BCL11A enhancer dissection by Cas9-mediated in situ saturating mutagenesis. *Nature*, *527*(7577), 192–197. https://doi.org/10.1038/nature15521

Casini, A., Olivieri, M., Petris, G., Montagna, C., Reginato, G., Maule, G., Lorenzin, F., Prandi, D., Romanel, A., Demichelis, F., & others. (2018). A highly specific SpCas9 variant is identified by in vivo screening in yeast. *Nature Biotechnology*, *36*(3), 265.

Chaim, L. H. K. T. (2014). *Interval Tree*.

Chaudhari, H. G., Penterman, J., Whitton, H. J., Spencer, S. J., Flanagan, N., Lei Zhang, M. C., Huang, E., Khedkar, A. S., Toomey, J. M., Shearer, C. A., Needham, A. W., Ho, T. W., Kulman, J. D., Cradick, T. J., & Kernytsky, A. (2020). Evaluation of Homology-Independent CRISPR-Cas9 Off-Target Assessment Methods. *The CRISPR Journal*, *3*(6), 440–453. https://doi.org/10.1089/crispr.2020.0053

Chen, J. S., Dagdas, Y. S., Kleinstiver, B. P., Welch, M. M., Sousa, A. A., Harrington, L. B., Sternberg, S. H., Joung, J. K., Yildiz, A., & Doudna, J. A. (2017). Enhanced proofreading governs CRISPR–Cas9 targeting accuracy. *Nature*, *550*(7676), 407.

Cong, L., Ran, F. A., Cox, D., Lin, S., Barretto, R., Habib, N., Hsu, P. D., Wu, X., Jiang, W., Marraffini, L. A., & others. (2013). Multiplex genome engineering using CRISPR/Cas systems. *Science*, *339*(6121), 819–823.

Consortium, 1000 Genomes Project, & others. (2015). A global reference for human genetic variation. *Nature*, *526*(7571), 68.

Consortium, E. P., & others. (2004). The ENCODE (ENCyclopedia of DNA elements) project. *Science*, *306*(5696), 636–640.

Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, *5*(1), 46–55.

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., & Durbin, R. (2011). The variant call format and VCFtools. *Bioinformatics*, *27*(15), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

D'haeseleer, P. (2006). What are DNA sequence motifs? *Nature Biotechnology*, *24*(4), 423–425. https://doi.org/10.1038/nbt0406-423

Doench, J. G., Fusi, N., Sullender, M., Hegde, M., Vaimberg, E. W., Donovan, K. F., Smith, I., Tothova, Z., Wilen, C., Orchard, R., & others. (2016a). Optimized sgRNA design to maximize activity and minimize off-target effects of CRISPR-Cas9. *Nature Biotechnology*, *34*(2), 184.

Doench, J. G., Fusi, N., Sullender, M., Hegde, M., Vaimberg, E. W., Donovan, K. F., Smith, I., Tothova, Z., Wilen, C., Orchard, R., & others. (2016b). Optimized sgRNA design to maximize activity and minimize off-target effects of CRISPR-Cas9. *Nature Biotechnology*, *34*(2), 184.

Doench, J. G., Hartenian, E., Graham, D. B., Tothova, Z., Hegde, M., Smith, I., Sullender, M., Ebert, B. L., Xavier, R. J., & Root, D. E. (2014). Rational design of highly active sgRNAs for CRISPR-Cas9–mediated gene inactivation. *Nature Biotechnology*, *32*(12), 1262.

FEUGHELMAN, M., LANGRIDGE, R., SEEDS, W. E., STOKES, A. R., WILSON, H. R., HOOPER, C. W., WILKINS, M. H. F., BARCLAY, R. K., & HAMILTON, L. D. (1955). Molecular Structure of Deoxyribose Nucleic Acid and Nucleoprotein. *Nature*, *175*(4463), 834–838. https://doi.org/10.1038/175834a0

Flores, M., Glusman, G., Brogaard, K., Price, N. D., & Hood, L. (2013). P4 medicine: how systems medicine will transform the healthcare sector and society. *Personalized Medicine*, *10*(6), 565–576. https://doi.org/10.2217/pme.13.57

Foster, M. W. (2008). Human Genome Diversity Project (<scp>HGDP</scp>). In *eLS*. Wiley. https://doi.org/10.1002/9780470015902.a0005173.pub2

Frankish, A., Diekhans, M., Ferreira, A.-M., Johnson, R., Jungreis, I., Loveland, J., Mudge, J. M., Sisu, C., Wright, J., Armstrong, J., Barnes, I., Berry, A., Bignell, A., Carbonell Sala, S., Chrast, J., Cunningham, F., Di Domenico, T., Donaldson, S., Fiddes, I. T., … Flicek, P. (2019). GENCODE reference annotation for the human and mouse genomes. *Nucleic Acids Research*, *47*(D1), D766–D773. https://doi.org/10.1093/nar/gky955

Fredkin, E. (1960). Trie memory. *Communications of the ACM*, *3*(9), 490–499. https://doi.org/10.1145/367390.367400

Fu, Y., Sander, J. D., Reyon, D., Cascio, V. M., & Joung, J. K. (2014). Improving CRISPR-Cas nuclease specificity using truncated guide RNAs. *Nature Biotechnology*, *32*(3), 279–284. https://doi.org/10.1038/nbt.2808

Fujimori, S., Davidson, B. L., Kelley, W. N., & Palella, T. D. (1989). Identification of a single nucleotide change in the hypoxanthine-guanine phosphoribosyltransferase gene (HPRTYale) responsible for Lesch-Nyhan syndrome. *Journal of Clinical Investigation*, *83*(1), 11–13. https://doi.org/10.1172/JCI113846

Haeussler, M., Schönig, K., Eckert, H., Eschstruth, A., Mianné, J., Renaud, J.-B., Schneider-Maunoury, S., Shkumatava, A., Teboul, L., Kent, J., & others. (2016). Evaluation of off-target and on-target scoring algorithms and integration into the guide RNA selection tool CRISPOR. *Genome Biology*, *17*(1), 148.

Herman, J. G., Merlo, A., Mao, L., Lapidus, R. G., Issa, J. P., Davidson, N. E., Sidransky, D., & Baylin, S. B. (1995). Inactivation of the CDKN2/p16/MTS1 gene is frequently associated with aberrant DNA methylation in all common human cancers. *Cancer Research*, *55*(20), 4525–4530.

Hume, A., & Sunday, D. (1991). Fast string searching. *Software: Practice and Experience*, *21*(11), 1221–1248. https://doi.org/10.1002/spe.4380211105

Jinek, M., Chylinski, K., Fonfara, I., Hauer, M., Doudna, J. A., & Charpentier, E. (2012). A Programmable Dual-RNA–Guided DNA Endonuclease in Adaptive Bacterial Immunity. *Science*, *337*(6096), 816–821. https://doi.org/10.1126/science.1225829

Johnson, A. D. (2010). An extended IUPAC nomenclature code for polymorphic nucleic acids. *Bioinformatics*, *26*(10), 1386–1389.

Karczewski, K. J., Francioli, L. C., Tiao, G., Cummings, B. B., Alföldi, J., Wang, Q., Collins, R. L., Laricchia, K. M., Ganna, A., Birnbaum, D. P., Gauthier, L. D., Brand, H., Solomonson, M., Watts, N. A., Rhodes, D., Singer-Berk, M., England, E. M., Seaby, E. G., Kosmicki, J. A., … MacArthur, D. G. (2020). The mutational constraint spectrum quantified from variation in 141,456 humans. *Nature*, *581*(7809), 434–443. https://doi.org/10.1038/s41586-020-2308-7

Karvelis, T., Gasiunas, G., Miksys, A., Barrangou, R., Horvath, P., & Siksnys, V. (2013). crRNA and tracrRNA guide Cas9-mediated DNA interference in *Streptococcus thermophilus*. *RNA Biology*, *10*(5), 841–851. https://doi.org/10.4161/rna.24203

Kent, W. J., Sugnet, C. W., Furey, T. S., Roskin, K. M., Pringle, T. H., Zahler, A. M., & Haussler, a. D. (2002). The Human Genome Browser at UCSC. *Genome Research*, *12*(6), 996–1006. https://doi.org/10.1101/gr.229102

Kleinstiver, B. P., Pattanayak, V., Prew, M. S., Tsai, S. Q., Nguyen, N. T., Zheng, Z., & Joung, J. K. (2016). High-fidelity CRISPR–Cas9 nucleases with no detectable genome-wide off-target effects. *Nature*, *529*(7587), 490–495. https://doi.org/10.1038/nature16526

Knuth, D. E., Morris, Jr. , J. H., & Pratt, V. R. (1977). Fast Pattern Matching in Strings. *SIAM Journal on Computing*, *6*(2), 323–350. https://doi.org/10.1137/0206024

Komor, A. C., Badran, A. H., & Liu, D. R. (2017). CRISPR-based technologies for the manipulation of eukaryotic genomes. *Cell*, *168*(1–2), 20–36.

Kreitman, M. (1983). Nucleotide polymorphism at the alcohol dehydrogenase locus of Drosophila melanogaster. *Nature*, *304*(5925), 412–417. https://doi.org/10.1038/304412a0

Kühn, R., Schwenk, F., Aguet, M., & Rajewsky, K. (1995). Inducible Gene Targeting in Mice. *Science*, *269*(5229), 1427–1429. https://doi.org/10.1126/science.7660125

Lessard, S., Francioli, L., Alfoldi, J., Tardif, J.-C., Ellinor, P. T., MacArthur, D. G., Lettre, G., Orkin, S. H., & Canver, M. C. (2017). Human genetic variation alters CRISPR-Cas9 on-and off-targeting specificity at therapeutically implicated loci. *Proceedings of the National Academy of Sciences*, 201714640.

Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, *25*(14), 1754–1760.

Lin, Y., Cradick, T. J., Brown, M. T., Deshmukh, H., Ranjan, P., Sarode, N., Wile, B. M., Vertino, P. M., Stewart, F. J., & Bao, G. (2014). CRISPR/Cas9 systems have off-target activity with insertions or deletions between target DNA and guide RNA sequences. *Nucleic Acids Research*, *42*(11), 7473–7485.

Lowy-Gallego, E., Fairley, S., Zheng-Bradley, X., Ruffier, M., Clarke, L., & Flicek, P. (2019). Variant calling on the GRCh38 assembly with the data from phase three of the 1000 Genomes Project. *Wellcome Open Research*, *4*, 50. https://doi.org/10.12688/wellcomeopenres.15126.2

Mali, P., Yang, L., Esvelt, K. M., Aach, J., Guell, M., DiCarlo, J. E., Norville, J. E., & Church, G. M. (2013). RNA-Guided Human Genome Engineering via Cas9. *Science*, *339*(6121), 823–826. https://doi.org/10.1126/science.1232033

Manber, U., & Myers, G. (1993). Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, *22*(5), 935–948. https://doi.org/10.1137/0222058

McKenna, A., & Shendure, J. (2018). FlashFry: a fast and flexible tool for large-scale CRISPR target design. *BMC Biology*, *16*(1), 74.

Montague, T. G., Cruz, J. M., Gagnon, J. A., Church, G. M., & Valen, E. (2014). CHOPCHOP: a CRISPR/Cas9 and TALEN web tool for genome editing. *Nucleic Acids Research*, *42*(W1), W401–W407.

Moore, J. E., Purcaro, M. J., Pratt, H. E., Epstein, C. B., Shoresh, N., Adrian, J., Kawli, T., Davis, C. A., Dobin, A., Kaul, R., Halow, J., van Nostrand, E. L., Freese, P., Gorkin, D. U., Shen, Y., He, Y., Mackiewicz, M., Pauli-Behn, F., Williams, B. A., … Weng, Z. (2020). Expanded encyclopaedias of DNA elements in the human and mouse genomes. *Nature*, *583*(7818), 699–710. https://doi.org/10.1038/s41586-020-2493-4

Munshi, A. (2009a). The opencl specification. *2009 IEEE Hot Chips 21 Symposium (HCS)*, 1–314.

Munshi, A. (2009b). The opencl specification. *2009 IEEE Hot Chips 21 Symposium (HCS)*, 1–314.

Nicholson, A. W. (2014). Ribonuclease III mechanisms of double-stranded RNA cleavage. *Wiley Interdisciplinary Reviews: RNA*, *5*(1), 31–48. https://doi.org/10.1002/wrna.1195

Perez, A. R., Pritykin, Y., Vidigal, J. A., Chhangawala, S., Zamparo, L., Leslie, C. S., & Ventura, A. (2017). GuideScan software for improved single and paired CRISPR guide RNA design. *Nature Biotechnology*, *35*(4), 347–349. https://doi.org/10.1038/nbt.3804

Pliatsika, V., & Rigoutsos, I. (2015). "Off-Spotter": very fast and exhaustive enumeration of genomic lookalikes for designing CRISPR/Cas guide RNAs. *Biology Direct*, *10*(1), 4.

Ran, F. A., Hsu, P. D., Wright, J., Agarwala, V., Scott, D. A., & Zhang, F. (2013). Genome engineering using the CRISPR-Cas9 system. *Nature Protocols*, *8*(11), 2281–2308. https://doi.org/10.1038/nprot.2013.143

Sander, J. D., & Joung, J. K. (2014). CRISPR-Cas systems for editing, regulating and targeting genomes. *Nature Biotechnology*, *32*(4), 347–355. https://doi.org/10.1038/nbt.2842

Sanjana, N. E., Shalem, O., & Zhang, F. (2014). Improved vectors and genome-wide libraries for CRISPR screening. *Nature Methods*, *11*(8), 783.

Scott, D. A., & Zhang, F. (2017). Implications of human genetic variation in CRISPR-based therapeutic genome editing. *Nature Medicine*, *23*(9), 1095.

Shalem, O., Sanjana, N. E., Hartenian, E., Shi, X., Scott, D. A., Mikkelsen, T. S., Heckl, D., Ebert, B. L., Root, D. E., Doench, J. G., & Zhang, F. (2014). Genome-Scale CRISPR-Cas9 Knockout Screening in Human Cells. *Science*, *343*(6166), 84–87. https://doi.org/10.1126/science.1247005

Slaymaker, I. M., Gao, L., Zetsche, B., Scott, D. A., Yan, W. X., & Zhang, F. (2016). Rationally engineered Cas9 nucleases with improved specificity. *Science*, *351*(6268), 84–88. https://doi.org/10.1126/science.aad5227

Stadtmauer, E. A., Fraietta, J. A., Davis, M. M., Cohen, A. D., Weber, K. L., Lancaster, E., Mangan, P. A., Kulikovskaya, I., Gupta, M., Chen, F., Tian, L., Gonzalez, V. E., Xu, J., Jung, I., Melenhorst, J. J., Plesa, G., Shea, J., Matlawski, T., Cervini, A., … June, C. H. (2020). CRISPR-engineered T cells in patients with refractory cancer. *Science*, *367*(6481). https://doi.org/10.1126/science.aba7365

Tsai, S. Q., Zheng, Z., Nguyen, N. T., Liebers, M., Topkar, V. v, Thapar, V., Wyvekens, N., Khayter, C., Iafrate, A. J., Le, L. P., Aryee, M. J., & Joung, J. K. (2015).

GUIDE-seq enables genome-wide profiling of off-target cleavage by CRISPR-Cas nucleases. *Nature Biotechnology*, *33*(2), 187–197. https://doi.org/10.1038/nbt.3117

Vakulskas, C. A., Dever, D. P., Rettig, G. R., Turk, R., Jacobi, A. M., Collingwood, M. A., Bode, N. M., McNeill, M. S., Yan, S., Camarena, J., & others. (2018). A high-fidelity Cas9 mutant delivered as a ribonucleoprotein complex enables efficient gene editing in human hematopoietic stem and progenitor cells. *Nature Medicine*, *24*(8), 1216.

Walton, R. T., Christie, K. A., Whittaker, M. N., & Kleinstiver, B. P. (2020). Unconstrained genome targeting with near-PAMless engineered CRISPR-Cas9 variants. *Science*, *368*(6488), 290–296. https://doi.org/10.1126/science.aba8853

Wiedenheft, B., Sternberg, S. H., & Doudna, J. A. (2012). RNA-guided genetic silencing systems in bacteria and archaea. *Nature*, *482*(7385), 331–338. https://doi.org/10.1038/nature10886

Zhu, L. J., Holmes, B. R., Aronin, N., & Brodsky, M. H. (2014). CRISPRseek: a bioconductor package to identify target-specific guide RNAs for CRISPR-Cas9 genome-editing systems. *PloS One*, *9*(9), e108424.