Ruprecht-Karls-University of Heidelberg
Faculty of Biochemistry

RARE: a framework for random region generation

Bachelor Thesis

by Elvira Mingazova

July 2017

National Center for Tumor Diseases (NCT) Heidelberg
Section Molecular and Gene Therapy

Supervisor:	Prof. Dr. Manfred Schmidt
Day-to-day supervisor:	Dr. Raffaele Fronza

The Bachelor Thesis was performed at the National Center for Tumor Diseases (NCT) Heidelberg in the Section Molecular and Gene Therapy in the period from 15.05.17 to 13.07.2017 under the supervision of Prof. Dr. Manfred Schmidt.

First Examiner:	Prof. Dr. Manfred Schmidt
Second Examiner:	Dr. Rainer Beck

I herewith declare that:

1. I wrote this Bachelor Thesis independently under supervision and that I used no other sources and supporting materials than those indicated;
2. the adoption of quotation from the literature/internet as well as thoughts from other authors were indicated in the Thesis;
3. my report was not submitted to any other examination.

I am aware of the fact that a false declaration will have legal consequences.

Heidelberg, 13^{th} July

Elvira Mingazova

Abstract

In the age of fast evolving gene therapy methods it is important to fairly asses the risks and possible side effects of such treatments. One of the ways to do so is to perform an integration site (IS) analysis of a particular vector type that involves a statistical comparison of an experimental data set and a random one, enabling to identify the preferential spots on the genome for the transgene insertion. The characteristics of the random dataset may strongly affect the results of an IS analysis, making it important to develop a suitable and flexible framework for such data set generation. In this Bachelor-Project we created three Random Models, that can generate a needed amount of IS on a genome of choice: RM^0 (the simplest model), RM^1 (mappability model) and RM^2 (enzyme bias model). We found out that the distributions of the positions generated by three distinct algorithms differ significantly, supporting the idea that a biologically correct random data set is not the same as the one that considered to be correct in terms of a computer-generated sequence. We believe that the IS analysis of the viral vectors has a potential to produce more true and precise results by using the relevant random data set.

Contents

1 Introduction

1.1 Viral vectors in gene therapy

Viruses introduce their DNA sequences into the host cell as part of their replication cycle and exploit the cellular machinery to facilitate their replication. Some of the viruses are able to incorporate their genetic material straight into the host DNA, others persist in the cell nucleus predominantly as extrachromosomal episomes. The ability of the viruses to deliver genetic material into the cell is applied in the gene therapy that aims to introduce a functional gene into a target cell and restore protein production that is absent or deficient due to a genetic disorder.

In general, a viral genome contains the coding regions and *cis*-regulatory elements (CREs). The coding sequences enclose the genetic information of the viral structural and regulatory proteins and are required for replication and propagation of infection, whereas CREs are essential for packaging the vector genome into the virus capsid and integration into the host cell. Over the years, a number of variously designed viral vectors have been developed. Ideal virus-based vectors for most gene-therapy applications preserve their ability to infect the host cell but avoid the subsequent expression of viral genes that leads to replication and toxicity. This is achieved by deleting all, or some, of the coding regions from the viral genome, but leaving intact the CREs. The transgenic expression cassette is then cloned into the viral backbone in place of the deleted sequences and so is the gene delivery vehicle ready to use for the gene therapy purposes [17, 30].

Several diseases have been successfully treated with gene therapy, including some haematological and metabolic disorders [6, 2, 11]. However, given the diversity of disease targets, there is no single vector suitable for all applications. There are five main classes of clinically applicable viral vector that are derived from oncoretroviruses, lentiviruses, adenoviruses, adeno-associated viruses (AAVs) and herpes viruses. Each of these classes of vector is characterized by a set of different properties that make it suitable for some applications and unsuitable for other. The only characteristics that are required by all vectors are the abilities to be reproducibly and stably propagated and purified to high titres, to mediate targeted delivery (that is, to deliver the transgene specifically to the tissue or organ of interest without widespread vector dissemination elsewhere) and to mediate gene delivery and transgene expression without inducing harmful side effects [30].

1.1.1 Insertional mutagenesis and safety issues

Over the past 10 years of clinical trials there were reported many successful gene therapy treatments of severe inherited diseases of the blood, immune and nervous systems, including primary immunodeficiencies, leukodystrophies, thalassaemia, haemophilia and retinal dystrophy, as well as cancers such as B-cell malignancies [23].

However, a viral vector mediated gene delivery is associated with certain risks. In fact, some clinical trials on animals and humans showed adverse effects after the viral vector integrations. One of the issues is the insertion of the vector into the wrong location of the host DNA that has a potential to alter the normal gene expression of the host and induce cancer in the worst case scenario. One study showed that the transplantation and expansion of a clone of retrovirally transduced bone-marrow cells had induced leukaemia in mice [18]. A similar failure has occured in an otherwise quite successful gene therapy trial treating human X-linked severe combined immunodeficiency (X-SCID). In three patients the gene-therapy vector has inserted near a proto-oncogene LIM domain only 2 (LMO2) and activated its transcription and in one patient two insertions were found, one in the LMO2 locus and the other in the B cell-specific Moloney murine leukaemia virus integration site 1 proto-ocogene (BMI1), so in total four of the twenty patients developed leukaemia [12, 13, 14, 27]. Another cancer-related gene, associated with integration sites in transformed cells that arise during human gene therapy, is MECOM (also known as MDS1 and EVI1 complex locus) [24, 27].

In total, there are only a few genes that were reported to be directly involved in cancer progression after gene therapy trials, but at the same time there are large databases on the cancer related genes arisen from different studies. For instance, historically, one of the main techniques for identifying genes that promote transformation has been surveying retroviral integration sites in tumours in model organisms. Cases in which integration sites are recovered repeatedly near the same gene in cancer cells but not in normal cells provide evidence for the involvement of these genes in transformation. Such studies have been carried out in mice and results were collected in the Retroviral Tagged Cancer Gene Database (RTCGD)[3]. Analysis showed that integration sites in tumours commonly lie near the starting point of transcription, either upstream or just within the transcription unit, often in a 5' intron. Proviruses at these locations usually increase the rate of transcription either via promoter or via enhancer insertion [16].

Still there is a lot to be done in order to understand what genomic regions can be considered as safe for the viral vector insertions, that is the genomic sites, where transgenes can be stably and reliably expressed in any tissue without harmful effects

on the endogenous gene structure and transcription. In the opinion-article by M. Sadelain et al. there are 5 criteria for genomic safe harbours proposed:

- Distance of > 50 kb from 5' end of any gene
- Distance of > 300 kb from cancer-related genes
- Distance of > 300 kb from any microRNA
- Outside a gene transcription unit
- Outside of ultra-conserved regions

These criteria aim to exclude the disruption of endogenous coding genes and ultra-conserved regions and to minimize the possibility of long-range interactions between vector-encoded transcriptional activators and the promoters of adjacent genes, particularly cancer-related and microRNA genes [27].

Even though the safety criteria are still to be further evolved, the ability to predict the integration landscape of a particular vector is of advantage when trying to assess mutational risks of the gene-therapy treatment. In fact, it is possible to analyse the pattern of integration of different viral vector types and to look for locations they prefer in various tissues. First, it was believed that retroviral vector genomes integrated randomly into host chromatin and the risk of inducing cancer was considered to be negligible taking into account the tiny proportion of the cancer-related genes in the full genome and the fact that oncogenesis usually requires multiple genetic lesions [30]. Later studies revealed, however, that the nature of integration of the most frequently used retroviral and lentiviral vectors is semi-random with preference of integrating in vicinity of the transcribed genes [4]. As mentioned above, the integration near or within cancer-related genes posed the greatest threat in the context of therapeutic cell engineering, making the avoidance of proximity to such genes a priority [22, 26].

The points listed above make it clear that the analysis of integration landscapes of a particular vector type and estimation of the risks associated with transgene insertions are of a high importance. In the next subsection of this thesis we will focus on the recent studies of the viral vector integration patterns and discuss the methods used to identify preferential locations for such integrations.

1.1.2 Integration site analysis

A lot of research has been already conducted in order to understand how exactly the viral vectors are incorporated into the host DNA. Modern technologies such as next generation sequencing (NGS) and linear amplification-mediated PCR (LAM-PCR) allow exact detection of the insertion cites and analysis of their side effects [28]. One comprehensive bioinformatic analysis of integration targeting was performed by Berry et al. in 2006. This study covered seven types of integrating elements (HIV, MLV, ASLV, SFV, L1,SB and AAV) and more than 200 types of genomic annotation, seeking for the new associations between genomic features and viral integration sites. The authors were looking at the distances between IS and numerous genomic features such as CpG Islands, DNaseI cleavage cites, transcription start and stop features etc., trying to detect whether an integrating element is favoring or disfavoring any of them compared to the random IS distribution.

The results were obtained as receiver operator characteristic (ROC) curves and summarized as color-coded heat maps, assessing the effects of genomic features on integration targeting by each element (see Figure 1.1). A red color on the heat map points that the area under the ROC curve is between 0.55 and 1, which, in turn, means that integration events of a particular element have higher values for the feature than a random control event. In other words, more integrations accumulate within the specific distance from a feature than expected in the random model. A green color points at the opposite behaviour: less integrating events observed within the distance from the genomic feature as expected in case of the random integration, corresponding to an area

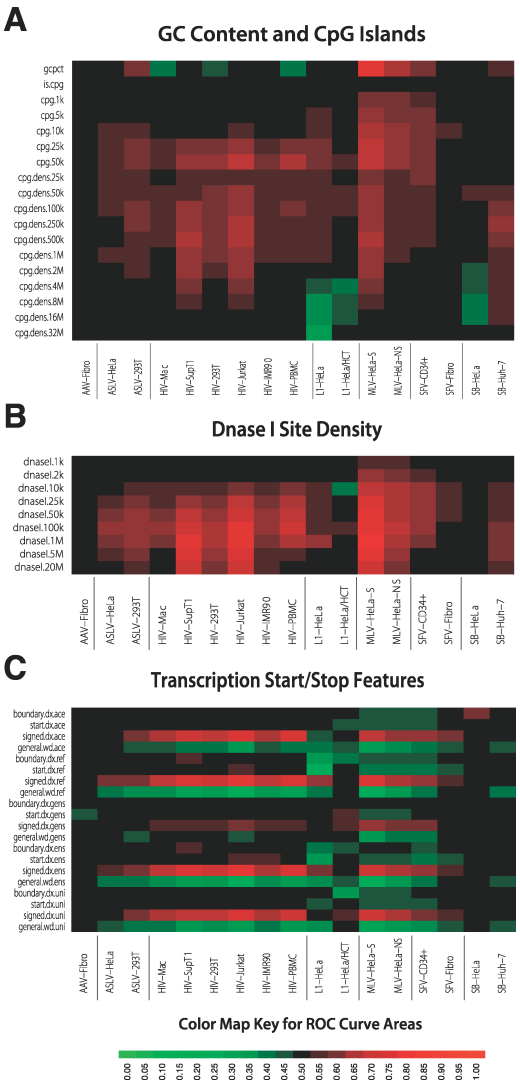


Figure 1.1: Example of the heat maps obtained in the ROC analysis by Berry et al. [4] (A) Effects of G/C content and CpG islands on integration
(B) Relationship between DNase I site density and integration frequency
(C) Effects of proximity to gene boundaries on integration.

under the ROC curve between 0 and 0.45. Finally, if there is no difference observed in the frequency of experimental and simulated random integrations, the space on the heat map is black. The effects of each feature were tested over intervals from 100 kb to 4 Mb.

The analysis of integration targeting has shown different results for each type of element. The AAV, for instance, demonstrated nearly random distribution of IS and no favoring towards transcription unit (TU) or gene 5' ends. In contrast, HIV was reported to favor active TUs and gene-dense regions. The MLV data sets showed a strong favoring of regions of high G/C for integration. Overall, all data sets were weakly positive for at least a few of the measures [4].

The above statements were possible to make after comparison of the experimental data with a random control data set, so it's quality plays a central role in the integration site analysis. This will be the main topic of discussion in the next chapters.

1.2 The role of the random data set in the integration site analysis

The results of the IS analysis depend on the random control data set that can be generated in a different ways. In the studies of integration landscapes of the viral vectors there is usually poor explanation provided on what is understood by a random position within a genome and how the random sample was obtained. In the paper from Berry et al. the following details are provided:

> For many of the datasets studied, it was possible to correct for possible biases by using a matched random control, in which each experimentally generated integration site was paired with ten random sites in the human genome that were constrained to lie the same number of base pairs from an appropriate restriction site. In the statistical analysis, each experimental integration site was compared with its matched random controls, thereby controlling for possible bias from restriction enzyme cleavage. For a few integration site datasets, it was not practical to generate matched random controls, so unmatched random sites were used[4].

These statements reveal that in some cases the authors were considering the fact that the experimental integration sites were cloned by methods, involving use of restriction enzymes to cleave the genomic DNA, flanking integrated elements. If this applied, the so-called *matched random controls* were used. In other cases, unmatched

random sites were generated, but there is no further explanation on the algorithm or mechanism of such a simulation.

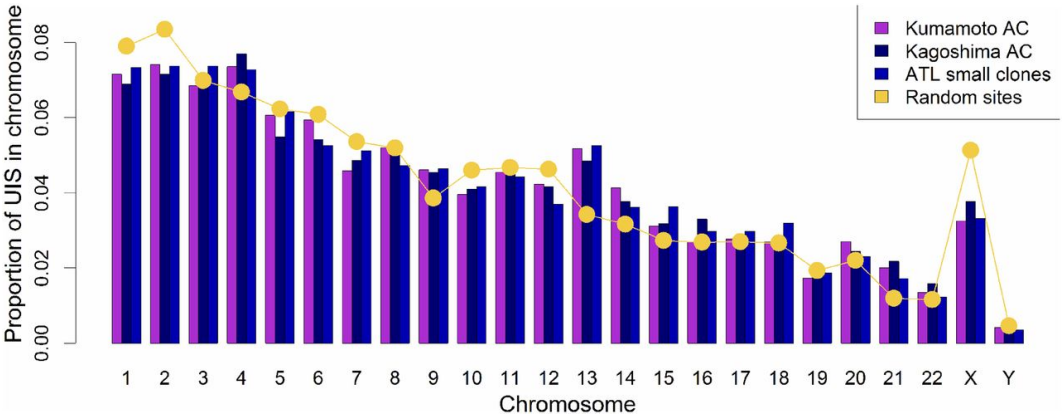


Figure 1.2: The proportion of unique integration sites (UIS) per chromosome: two independent asymptomatic HTLV-1 carriers (AC) data sets (Kumamoto and Kagoshima) and the small clones in adult T-cell leukaemia/lymphoma (ATL) cases, the yellow line shows the frequency of sites in the random data set. An increased number of integrations in chromosomes 13, 14, 15, and 21 in the clones of asymptomatic carriers and small clones in ATL cases was observed, compared with the random data set[7].

In a different study of the HTLV-1 genomic integration conducted by Cook et al. there are many mentions of the random data set, but also no details on it's kind. The only remark is that they used the DNA shearing to detect the proviruses, which doesn't involve restriction enzymes, so there was no need to correct for the bias in the random data set [7]. Presumably, the simplest random model was used which selected in each round a chromosome according to it's relative size and generated uniformly distributed positions within each chromosome. In Figure 1.2 taken from Cook et al. three experimental data sets are plotted against the random sites. On average, the HTLV-1 integration pattern is very similar to the random one, the authors claim, however, that the number of integrations in chromosomes 13, 14, 15 and 21 in the experimental data sets are significantly increased compared to random [7].

In this bachelor project we will develop our own framework for random data set generation suitable for different purposes. Later in this thesis we will test, if we can reproduce the results for the HTLV-1 integration sites using this new framework.

1.3 Purpose and stages of the framework developement

This bachelor thesis was part of a bigger project of IS analysis similar to the one conducted by Berry et al. Due to the fact, that there was no overall accepted and thoroughly explained strategy for the random data set generation in the research papers over the past 15 years, we decided to develop a computer-based framework for random region generation (RARE), that would enable us to create a random data set suitable for any purpose with high biological accuracy. These ideas will be presented in more detail in the next sections explaining the concepts of mappability and enzyme bias incorporated into RARE.

As mentioned before, the random model that is generating control IS may strongly affect the results produced by IS analysis. Simple generation of random positions anywhere on the genome of interest appears to be biologically false, since the experimental data points to be compared with a random dataset are impossible to detect on the unmappable parts of the genome and, in some cases, in the regions far away from the restriction enzyme cleavage sites. The latter applies for the LAM-PCR based sequencing and detection of IS [28, 8, 4]. Hence this project was split into three parts, corresponding to the random models that were developed, starting from the simplest one as a basis:

1. Random Model Zero (RM^0): unconditional random IS generation
2. Random Model One (RM^1): mappability
3. Random Model Two (RM^2): enzyme bias

Each of the random models is able to generate a dataset within a genome of choice. Depending on the conditions and purposes of the experiment, one or another model may suit better. This bachelor thesis includes a detailed description of the development stages of the RARE framework, the concepts it is based on, materials and methods used to program the simulator and comparison of the datasets generated by each of the random models.

1.3.1 Random Model Zero and the concept of randomness

First model represents the simplest random region generator on the genome of choice. At this point it is important to define, what is a random number in terms of a computer based generation. One of the accepted definitions of randomness in the context of computer-generated sequences, is to say that the deterministic program that produces a random sequence should be different from, and statistically uncor-

related with, the computer program that uses it's output. In other words, any two different random number generators ought to produce statistically the same results when coupled to a particular application program. There is a list of statistical tests, which can find out any correlations that are likely to be detected by an applications program. A good random number generator ought to pass all of these tests [25].

Another important term for this thesis is a definition of a *simple random sample*. A simple random sample is a subset of individuals (a sample) chosen from a larger set (a population). Each individual is chosen randomly and entirely by chance, such that each individual has the same probability of being chosen at any stage during the sampling process, and each subset of k individuals has the same probability of being chosen for the sample as any other subset of k individuals [29]. If applying this definition to the generation of random positions on a given chromosome, it can be reduced to the following idea:

> A random position on a chromosome c is a random number in the interval $[1, len(c)]$, where $len(c)$ is the length of the chromosome c. The random number is generated using a uniform distribution model.

So each position on a chromosome c has the same probability to be chosen at each round of a random number generation, accordingly, the distribution of random positions generated with RM^0 is expected to be uniform within one chromosome (see Figure 3.5 for proof). However, among all the chromosomes of the genome the distribution is expected to correlate with their relative sizes, which can be achieved upon weighted choice of the chromosome, that is, the bigger chromosome has to be chosen more often than the smaller one (see section 2.1). This idea is well visualized on Figure 1.2, where the proportion of random IS in the respective chromosome goes along with its size.

1.3.2 Random Model One and the concept of mappability

In the IS analysis real positions of the viral DNA-sequences within a host genome are compared with the artificial, usually, *in silico* generated random data set. At this point it is important to think about the experimentally detected IS and ask a question: Is it possible to retrieve all the integration spots of the vector within a genome or is some information lost during the process of retrieval? The current answer is negative, and the first reason lies in the characteristics of the sequencing technics.

Genome sequencing technologies (HTS - high throughput screening) produce huge amounts of short sequences (reads), with different lengths, error rates etc.,

depending on the technology employed. After completing the sequencing itself those short reads need to be aligned (mapped) onto a reference genome. The genome, however, has a nonrandom nature and includes a significant proportion of repetitive sequences, so one needs to distinguish between uniquely mapping reads (can be aligned to only one single location) and multiply mapping reads (match multiple possible locations). For a given genome, the proportion of uniquely mapped reads depends mostly on the length of the sequence reads produced by the experiment, and on the number of mismatches allowed during the mapping step [8].

In the detection of IS the mapping step of the virus vector flanking genomic sequence is always followed by filtering of the bad quality reads that give multiple hits in the genome. A configurable threshold is usually applied for allowing mismatches. Depending on the IS retrieval pipeline, the multiply mapped reads may be discarded [1]. For example, if the GEM algorithm is used for the reads alignment and 'unique-mapping' mode is chosen, then the reads having only one exhaustive match in the reference are aligned and the remaining reads are flagged as multiply mapping and are removed [20]. Hence, if the integration of the viral vector occurred within a repetitive region of the genome, its position will not be detected by the pipeline.

The idea of the *RM*[1] is to restrict the random region generation to only uniquely mappable parts of the genome and make the random control more suitable for the comparison with an experimental data set. In fact, given the technical specifications of the sequencing and IS detection experiment it is possible to compute *a priori* the mappability of the whole sequence, i.e., the inverse of the number of times that a read originating from any position in the reference genome maps back unambiguously to the genome itself - thus identifying the regions that are truly mappable. In the paper by Derrien et al. the following definition of mappability is provided:

> Given some read length k, the k-frequency $F_k(x)$ of a sequence at a given position x corresponds to the number of times the k-mer starting at position x appears in the sequence and in its reverse complement, considering as equivalent all the k-mers which differ by less than some predefined alignment score (e.g. a number of allowed mismatches). The analogous quantity, the $k - mappability$ or $k - uniqueness$ $M_k(x)$, is defined as the inverse of the frequency: $M_k(x) = 1/F_k(x)$ [8].

For each position in the genome it is possible to compute the $k - mappability$ and restrict the random model to select sites only within the regions where $k - mappability = 1$ (the k-mer found only once in the genome up to the specified number of mismatches). The mappability value is a quantity between 0 and 1. This fact contributes to a good visualization compared to the frequency value, that can span too large range from 1 to millions. For the exact tool and algorithm used to

compute the mappability see section 2.2.1.

1.3.3 Random Model Two and the concept of enzyme bias

In the previous subsection we explained why some of the integration cites are impossible to detect - because they may lay within a highly repetitive region of the genome. In this subsection we will explore another constrain produced by the technique widely used for identifying integrated proviral and host genomic DNA junctions - LAM-PCR. This technique relies on the presence of the restriction enzyme cutting site adjacent to a retrievable integration site. The schematic mechanism of the method is presented in Figure 1.3.

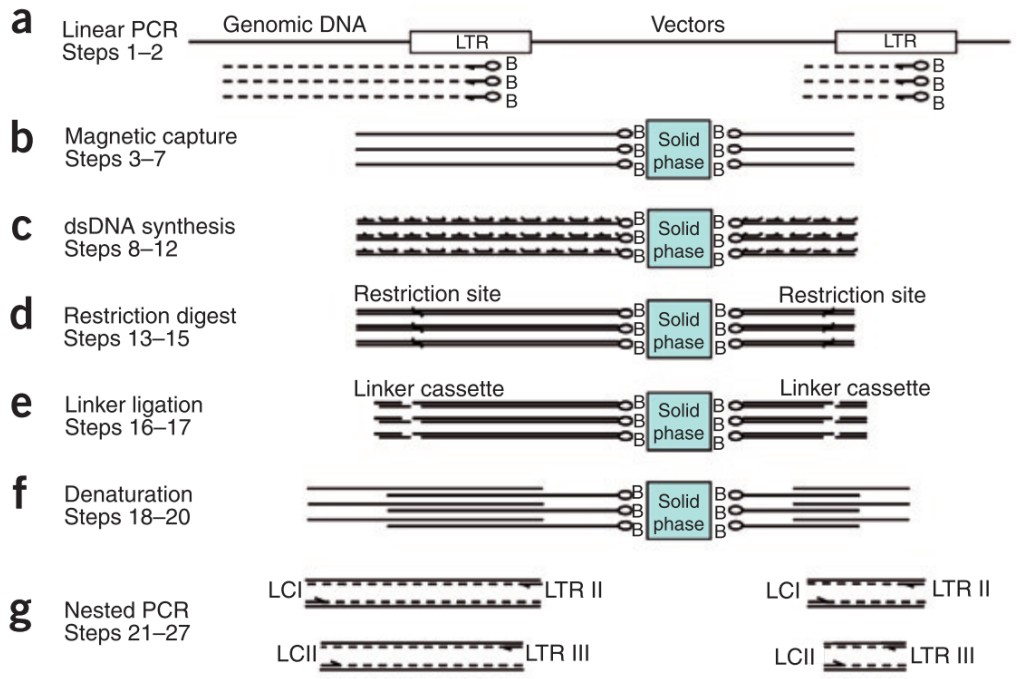


Figure 1.3: Schematic outline of LAM-PCR to amplify 5'-LTR retroviral vector-genomic fusion sequences [28]

First, the biotinylated target DNA is preamplified and captured on the paramagnetic beads Then it is digested with a single or a set of restriction enzymes and ligated to a restriction site complementary linker cassette (LC). The ligation product is then amplified by two nested PCR with primers specific for the vector long terminal repeat (LTR) and the LC sequences. As a result, two fragments are

generated from each integration clone: the vector-genome junction of an integration clone as well as the internal vector sequence.

Schmidt et al. recommend the usage of 4-cutter restriction endonucleases [28]. An advantage of using them is a higher frequency of cuts (compared to the 5- or 6-cutters) that can be withdrawn from a simple probability calculation: the chance of finding a certain base at a specific position is $\frac{1}{4}$, the probability to find the specific sequence of 4 bases (e.g. TTAA) is $(\frac{1}{4})^4 = \frac{1}{256}$, so every 256 bp a cleavage site is expected to be found by chance in case of a random sequence of bases. In reality, the sequence of bases has a non-random nature and the cleavage sites are not distributed evenly. Thus there are regions where there is no restriction site to be found, which results in longer genomic DNA fragments and, as short fragments are favoured [5], lower efficiency amplification, leading to loss of the IS in this fragments. This problem can be minimized by repeating the procedure with another enzyme, reducing the bias, which, in turn, will inevitably result in an increased time consumption [10].

There are other methods, evolving in the present, that allow to avoid the use of restriction enzymes in the IS detection, such as non-restrictive LAM-PCR (nrLAM-PCR)[9, 10] and restriction enzyme-free LAM-PCR (Re-free LAM-PCR) [31]. It would be an improvement if those methods would allow more complete and precise IS retrieval. However, the original LAM-PCR remains nowadays the standard well-established and widely used procedure. Moreover, the historical data sets, that were analysed using LAM-PCR, will probably not be reanalyzed using new methods, but are still interesting for further investigations; hence the problem of enzyme bias is to be considered important in the random sample generation.

Implementation of this idea is based on the pre-scanning of the genome and discovery of restriction enzyme cleavage sites within it. The random position generation is then matched to the cutting sites found, avoiding the regions far away from them. This helps to imitate the LAM-PCR retrievable IS.

2 Materials and Methods

This section provides a detailed explanation of the methods, materials, tools and algorithms used to develop each of the random models (RM^0, RM^1 and RM^2).

Implementation of the code for all three random models was based on:

Programming languages: Python (version 2.7.13), Bash Shell

Web application for writing and testing the code: Jupyter Notebook[1]

Code repository site: GitHub[2]

Server: OpenStack

2.1 Random Model Zero (RM^0)

For the simplest random region generation on a genome the python program has to be provided with an input file containing dimensions of the genome of choice. The code is implemented in a way that it, first, reads in the dimension file (see Table 2.1), then creates a dictionary with chromosome names as keys and chromosome lengths as values and, second, generates a random position/region on the randomly chosen chromosome. The chromosome choice is weighted according to it's size. the probability to choose each chromosome is:

$$P = \frac{len(chr)}{len(genome)} \tag{2.1}$$

Position/region generation is repeated n times, where n represents the number of random regions to be generated and is one of the parameters passed to the program by the user. The other parameter r (*range*) specifies possible lengths of the region and d (*delta*) expands the range to a certain chosen value.

The simulator provides the results in the BED format, see figure 2.1. The script `getRegionW2.py` (see Suplemental, page 49) was tested with three genomes: human[3]

[1] Allows to create and share documents that contain live code, equations, visualizations and explanatory text; `http://jupyter.org`

[2] All the scripts, data and figures are stored here `https://github.com/ElviraMingazova/RARE`

[3] `http://genome.ucsc.edu/cgi-bin/hgTracks?db=hg38&chromInfoPage=`

Table 2.1: Input file format for RM^0.

Chromosome name	length in bp
chr1	248956422
chr2	242193529
chr3	198295559
...	...
chrY	57227415

```
!python2 ../scripts/getRegionW2.py -n5 -r0 -d1 -i "../data/hg38.txt"
Checking format: the file is well formatted
Checking the first column: the names of chromosomes are correct
Checking the second column: the lengths of chromosomes are correct
chr#    Start           End             rnd#
chr19   8699793         8699794         rnd1
chr18   27883540        27883541        rnd2
chr14   79934262        79934263        rnd3
chr14   42162829        42162830        rnd4
chr7    82799459        82799460        rnd5
```

Figure 2.1: Example input and output of the RM^0 in the Jupyter Notebook. The command above reads in the dimension file `hg38.txt`, checks the format and generates 5 random positions on the hg38 assembly of the human genome.

(hg38), mouse (mm10)[1] and honey bee (apiMel2)[2]. Dimensions of the respective genomes were downloaded from the database of the University of California, Santa Cruz (UCSC).

2.2 Random Model One (RM^1)

2.2.1 Computation and visualization of mappability

The mappability of the genome of choice was computed using the GEM library tools[3], the algorithm to compute the mappability is described in the paper by Derrien at al. [8].

[1] `http://genome.ucsc.edu/cgi-bin/hgTracks?db=mm10&chromInfoPage=`

[2] `http://genome.ucsc.edu/cgi-bin/hgTracks?db=apiMel2&chromInfoPage=`

[3] `http://algorithms.cnag.cat/wiki/The_GEM_library`; Tools were downloaded from `https://sourceforge.net/projects/gemlibrary/files/gem-library/Binary%20pre-release%202/`

The following parameters were chosen:

- -T 8 (number of threads $T = 8$)
- -l k-mer (length of the k-mer of choice)
- -m 0.04 (default number of mismatches $m = 0.04$)

  The chosen value allows 4% of the k-mer length to be not corresponding with the reference sequence. The more mismatches are allowed, the less mappable the genome will appear. The method produces the exact mappability in the case of 0 mismatches, but mismatches are often allowed when mapping HTS reads at the step of the IS detection [1], so we also allowed mismatches in the mappability computation.

- -t "disabled" (per default approximation parameter t is disabled)

  The approximation parameter helps to optimize the computation time. For example, the parameter t is set to seven: if the k-mer matches some positions in the genome more than seven times ($F_k(x)$), all its occurrences are treated as already mapped and the frequency value is assigned to all of the positions, where the k-mer matches, so they are can be altogether skipped as the computation moves further along the genome. We decided to use no approximation parameter in order to compute the exact mappability, that resulted in somewhat longer computation time ($\sim$ 20 min for the chrY, $\sim$ 8 h for the complete human genome hg38).

To test the tools yeast and human chromosome Y sequences were downloaded from the UCSC server[1]. The right genome file format to compute the mappability is FASTA, for this reason the yeast genome file was converted to the FASTA format using `twoBitToFa`[2] tool.

Mappability was computed using the following algorithm:

1. Create an indexed genome using `gem-indexer` with the default parameters
2. Compute mappability for several k-mer lengths:
   a) Create an array/list of k-mer lengths (For the yeast k-mer = $[30, 150, 500]$

[1]Yeast: `http://hgdownload.soe.ucsc.edu/goldenPath/sacCer3/bigZips/sacCer3.2bit` ChrY: `http://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes/chrY.fa.gz`

[2]Downloaded from the utilities directory of UCSC `http://hgdownload.soe.ucsc.edu/admin/exe/`

were used, for the human chrY k-mer = $[30, 70, 150]$)

b) For each k-mer in an array of k-mer lengths compute the mappability using `gem-mappability` and write the output into a respectively named file

3. Convert results to the BigWig format appropriate for visualization. Use, first, `gem-2-wig` and then `wigToBigWig` converter

The same algorithm was used to compute the mappability of the human genome. Visualization of the mappability track was performed in the free accessible Integrative Genomics Viewer (IGV)[1], see section 3.1 for the results.

To prove the correctness of the results obtained for the yeast genome with the GEM library tools they were compared with the BLAST results at the two randomly chosen positions:

- Chromosome I, position 140008 (Gene: SSA1 YAL005C)
- Chromosome XII, position 469317 (Gene: ASP3-1 YLR155C)

The respective sequences of 30, 150 and 500 bp were cut staring from each of the randomly chosen positions[2] and aligned to the yeast genome[3] using the Standard Nucleotide BLAST[4] with a chosen megablast algorithm for the highly similar sequences.

For the human chromosome Y, mappability track gaps positions (scaffolds, heterochromatin, telomeres) were compared with the ones listed in the UCSC table browser[5], accessible through choosing of the following options:

For the results see section 3.2.

[1] `http://software.broadinstitute.org/software/igv/`

[2] The sequences were downloaded here http://fungi.ensembl.org/Saccharomyces_cerevisiae/Gene/Sequence?db=core;g=YAL005C;r=I:139503-141431;t=YAL005C and here `http://fungi.ensembl.org/Saccharomyces_cerevisiae/Gene/Summary?db=core;g=YLR155C;r=XII:469317-470405;t=YLR155C`

[3] Saccharomyces cerevisiae (taxid:4932)

[4] `https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastn&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome`

[5] `http://genome.ucsc.edu/cgi-bin/hgTables`

Table 2.2: UCSC table browser options to find out positions of gaps on the human chrY.

genome	human
assembly	hg38
position	chrY:1-57227415
group	mapping and sequencing
track	gap
output format	all fields from selected table

2.2.2 Uniquely mappable regions of the human genome

The mappability of the hg38 for the three k-mer lengths $[40, 100, 200]$ was computed using the algorithm described in the section 2.2.1. The next step of the framework development was to compute the percentage of the uniquely mappable regions on each chromosome (see section 3.3 for the results). For this purpose it was looped over all the positions (x) on a respective chromosome and if the mappability value ($M_k(x)$) was equal to one, the counter variable ($lenUM$) was increased by one.

$$M_k(x) = \begin{cases} 1, & lenUM+ = 1 \\ < 1, & \text{continue} \end{cases} \tag{2.2}$$

Here is the sequence of steps performed to calculate the results:

1. Convert WIG format files to the BED format using `wig2bed` function from the BEDOPS package[1]: After this procedure the mappability file contains 5 columns: chromosome name, starting position, end position, index and the mappability value between 0 and 1.

```
chr1    0       10000   id-1    0.000000
chr1    10000   10001   id-2    0.006463
chr1    10001   10005   id-3    0.005249
chr1    10005   10007   id-4    0.006463
chr1    10007   10011   id-5    0.005249
...     ...     ...     ...     ...
```

2. Extract a text file with two columns where the first column is a chromosome name and the second column is the length of the uniquely mappable parts:

   a) Create an array/list of chromosome names

[1] `http://bedops.readthedocs.io/en/latest/content/reference/file-management/conversion/wig2bed.html`

b) For each chromosome in a list of chromosomes extract the lines corresponding to the respective chromosome, iterate over each line and count how many positions are uniquely mappable

c) For each chromosome append the results to a file with a name corresponding to the k-mer length

To select the lines corresponding to the specific chromosome the `bedextract` command[1] from BEDOPS tools was used. The output file contains the information needed to compute the percentage of the uniquely mappable regions of the human genome for the respective k-mer length (see Tables 3.5, 3.6 and 3.7)

```
chr1    183895114
chr2    200761632
chr3    167230863
chr4    158680130
chr5    148972197
...     ...
```

2.2.3 Generation of the random dataset on the uniquely mappable regions of the human genome

This subsection provides an algorithm for the random IS generation starting from the precomputed mappability files in BED format. Important difference compared to the RM^0 is not only restriction of the regions IS can be generated on to only mappable ones, but also that the position generation on each chromosome is weighted by it's mappability percentage. This idea is presented in the equation 2.3

$$n_{ci} = N \frac{M_{ci}}{TM}, \tag{2.3}$$

where N is the total number of positions (IS) to be generated on the genome, n_{ci} is the number of positions to be generated on the respective chromosome i, M_{ci} (*Mappability of the chromosome* i in bp) is the sum of all the positions on the respective chromosome where the mappability value is equal to one, and TM (*Total mappability*) is the length of the genome with the mappability value equal to one. Total mappability is a sum of the uniquely mappable regions over all the

[1] `http://bedops.readthedocs.io/en/latest/content/reference/set-operations/bedextract.html`

chromosomes of the genome:

$$\sum_{i=1}^{C} M_{ci} = TM, \tag{2.4}$$

where C is the total number of chromosomes.

To generate N positions on hg38 using the respective k-mer map, the following steps were taken:

1. From the precomputed mappability files in BED format extract for each chromosome two columns: positions on a specific chromosome and True/False statement according to the mappability value. If the region is uniquely mappable ($M_c(x) = 1$) then "True" should be written opposite to it's starting position and if $M_c(x) < 1$ - "False". The unmappable regions following each other are combined. An extracted text file has the following format:

```
0 false
10314 true
10315 false
10596 true
10611 false
...    ...
```

2. Load the file into a two-dimensional (2D) numpy array in python and generate N positions on the mappable regions of hg38 using the script provided in the Supplemental (see page 53). The random dataset generated with RM^1 has the following format:

Chr	Start	End	Match	RM	Assembly
chr1	52387414	52387415	unmatched	map_40	hg38
chr1	115740335	115740336	unmatched	map_40	hg38
chr1	150974849	150974850	unmatched	map_40	hg38
chr1	40061190	40061191	unmatched	map_40	hg38
chr1	160125588	160125589	unmatched	map_40	hg38
...	...	...	...	...	...

2.3 Random Model Two (RM^2)

This model is based on the previous one (RM^1) with the difference that it generates random positions matched to the precomputed restriction enzyme cleavage sites and tests whether each position lays inside of the mappable region keeping only those that pass the test. To generate a random dataset using this logic the following

steps were taken:

1. Choose the restriction enzyme corresponding to the one used for the experimental IS retrieval. During this project datasets matched to four restriction enzymes were generated: MseI, Tsp 509, PstI, AflII.

Table 2.3: Restriction enzymes cleavage sites

Enzyme	Recognition pattern
MseI	5'... TTAA ...3'
Tsp509	5'... AATT ...3'
PstI	5'... CTGCAG ...3'
AflII	5'... CTTAAG ...3'

2. Extract the starting positions of the cleavage sites for a particular enzyme using the tool `scan_for_matches`[1]

   a) Create a pattern file containing the recognition sequence[2]

   b) Find the coordinates of this pattern on the genome using `scan_for_matches` and the pattern file for the enzyme of choice, select only the first column. The output file has the following format:

   ```
   chr1:[11533,11536]:
   chr1:[11556,11559]:
   ...
   ```

   c) For each chromosome extract all the starting positions of the restriction sites and write them into a separate file: The output file contains only one column with the restriction site start coordinates:

   ```
   100000132
   100000416
   100000561
   100000636
   100000860
   ...
   ```

3. Load the file into a one-dimensional (1D) numpy array in python and generate

[1] Downloaded from `http://blog.theseed.org/servers/2010/07/scan-for-matches.html`

[2] The recognition sequences were looked up here:
`https://www.neb.com/tools-and-resources/interactive-tools/enzyme-finder?searchType=7&recognitionSite=&matchType=1`

N positions on the mappable regions of hg38 matched to the respective enzyme restriction sites; the script is provided in the Supplemental (see page 55).

The output file has the same format as generated with RM^1 with the difference that the dataset is matched to an enzyme's restriction sites.

```
Chr     Start          End            Match   RM      Assembly
chr1    85364380       85364381       MseI    map_40  hg38
chr1    91532644       91532645       MseI    map_40  hg38
chr1    218161117      218161118      MseI    map_40  hg38
chr1    223049867      223049868      MseI    map_40  hg38
chr1    174190581      174190582      MseI    map_40  hg38
...     ...            ...            ...     ...     ...
```

2.4 Statistical tests

2.4.1 Binomial test

Binomial test was applied to investigate the distribution functions of positions on chr1 of the human genome, generated with three Random models. The purpose was to evaluate the frequency in each bin of the particular histogram and decide if this frequency was possible to obtain assuming a random (uniform) distribution under a chosen level of significance (see section 3.4). The test was performed in Python using the SciPy module[1]

Example: in the RM^0 distribution of 10000 positions on the chr1 divided into 100 bins we expect 100 positions in each bin on average, in case the data points are distributed uniformly. In the first bin we find 82 positions. Is it significantly different than would be expected by chance, on the null hypothesis of a random distribution? The Binomial distribution is used to calculate the probability of 82 or more extreme result (two-sided test): `scipy.stats.binom_test(82,10000,0.01)` command returns the P-value of 0.070, which is still bigger than the standardly used significance level of 5%, so we cannot reject the null hypothesis that this number is drawn from the uniform distribution.

The bin width choice was based on the square root rule $k = \sqrt{n}$, where k is number of bins and n is the sample size. In addition, the Bonferroni correction was applied[2]; it is used when the multiple-comparison takes place. While the α value is

[1] `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.binom_test.html`

[2] `http://mathworld.wolfram.com/BonferroniCorrection.html`

appropriate for each individual comparison, it is not for the set of all comparisons. Upon correction the test is passed when $P < \frac{\alpha}{n}$, where n is the number of tests performed.

In case of the *RM1* and *RM2* the expected number of data points in each bin had to be corrected, since the models restricted the selection of positions to only uniquely mappable regions. The total number of bins (B) spanning the unmappable region of chr1 is a fraction of its length (L_{unm}) and the bin size(S_b):

$$B = \frac{L_{unm}}{S_b} \tag{2.5}$$

The computation showed that ∼26 bins span the unmappable region; therefore recalculated number of expected data points in each bin under the null hypothesis is ∼135.

2.4.2 Kolmogorov-Smirnov test

The two-sample Kolmogorov-Smirnov test (K-S test) was implemented to pair-wise compare the datasets generated by the three random models. The question that it answers is: Are the two sets drawn from the same distribution function?

For K-S test the list of data points is converted to an unbiased estimator $S_N(x)$ of the Cumulative distribution function (CDF) of the probability distribution from which it was drawn. In the case of the two-sample K-S test, CDFs of both datasets are used to calculate the Kolmogorov-Smirnov D-statistics. The D is defined as the *maximum value* of the absolute difference (distance) between two CDFs. For comparing two different cumulative distribution functions $S_{N1}(x)$ and $S_{N2}(x)$, the K-S statistic is

$$D = \max_{-\infty<x<+\infty} |S_{N1}(x) - S_{N2}(x)| \tag{2.6}$$

The P-value is calculated based on the distribution of the D-statistics under null hypothesis. The P- value is defined as the probability of obtaining a result equal to or more extreme than what was observed under null hypothesis. The null hypothesis assumes, that there is no significant difference between two datasets and they belong to the same population. The null hypothesis is disproved or rejected if the P - value is less than the chosen significance level α (standard $\alpha = 0.05$), so the alternative hypothesis is accepted. If the P-value is higher than α, there is no evidence to reject the null hypothesis.[25]

In this project a simple `ks_2samp`[1] function from the `scipy.stats` module was used to compare pairwise datasets generated with the random models and the `hist`[2] function from the `matplotlib.pyplot` module was used to visualize the CDFs of the random datasets (see Figure 3.8).

[1] `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.ks_2samp.html`

[2] `https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.hist.html`

3 Results and discussion

This chapter includes visualizations of the results produced during the framework development. We will start directly with the Random Model One RM[1] and shift the comparison of the data set distributions towards the end of this section after the mappability is fully explored and discussed. Furthermore, we will test the random character of each of the three distributions generated, and, finally, look if we can reproduce the results of the recent research paper by Cook et al. [7] on the HTLV-1 integration landscape and investigate potential differencies.

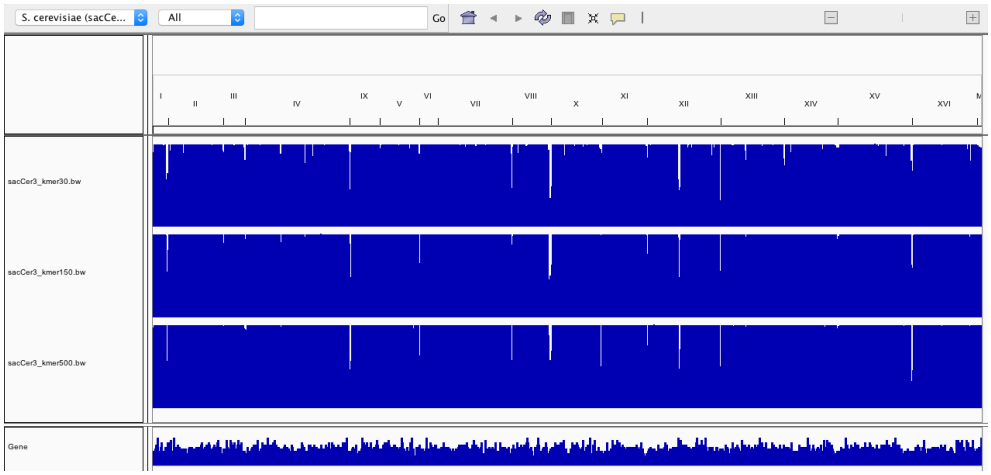
3.1 Visualization of mappability: yeast and human chrY

Mappability computation and visualization was a crucial step in the development of the RARE. It offered a basis for the Random Models One and Two, both being restricted to generate random positions only within the precomputed uniquely mappable regions of the genome of choice.

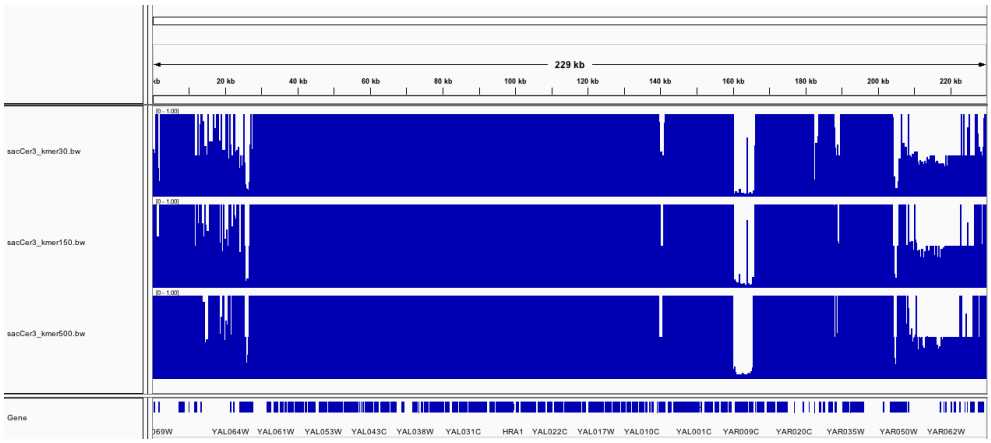
Before computing the mappability for the full human genome (hg38), the GEM library tools [8] were tested on the relative short DNA sequences of the yeast and human chromosome Y (assembly hg38) (chrY) (version hg38). In total, the mappability of this two sequences was computed three times each, corresponding to the k-mer lenghts (Yeast: k-mer $= [30, 150, 500]$ human chrY k-mer $= [30, 70, 150]$) and took less than half an hour in the first and less than one hour in the second case. We were expecting a very good mappability for the yeast genome, due to the high proportion of gene count (about 6000 genes [19]) in the total sequence of $\sim$12 Mb distributed among 16 chromosomes[1], and a rather bad mappability of the human chrY, since it is gene poor (519 genes in the total sequence of $\sim$57 Mb including pseudogenes shown on the BioMart database[2]) and reach in repetitive sequences (satellite DNA) and heterochromatic regions [15].

In Figure 3.1 there are three subfigures visualizing the mappability of the full genome of the yeast (3.1a), zoomed in chrI (3.1b) and chrXII (3.1c) tracks; each subfigure includes a line with the coordinates, mappability tracks corresponding to the k-mer lengths and the gene density line that contains the names of the genes, when zoomed in. The level of the blue color corresponds to the mappability value between 0 (white area) and 1 (fully filled with blue). The tracks reveal a very good mappability level on average, even for the shortest k-mer of 30 bp. In fact, the

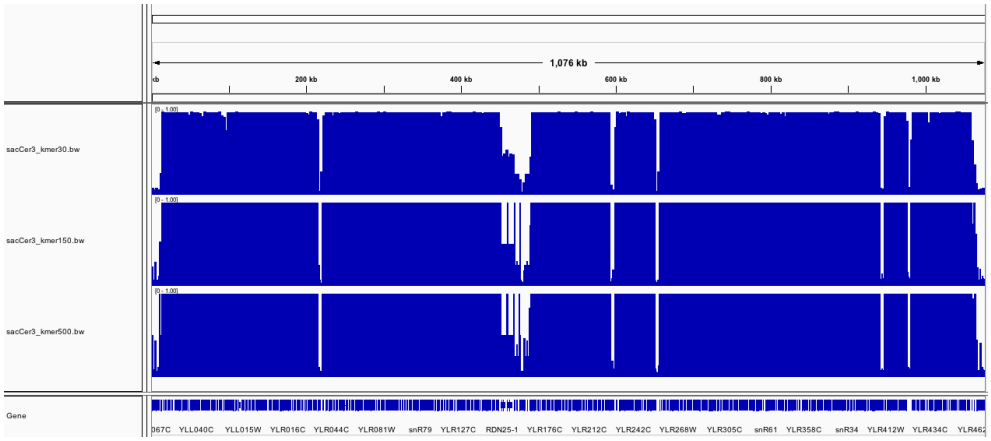
[1] `https://www.ncbi.nlm.nih.gov/genome?term=saccharomyces%20cerevisiae`
[2] `http://www.ensembl.org/biomart/martview/88cc691f0659b88601b428411b42f6e4`



(a) Full genome



(b) ChrI



(c) ChrXII

Figure 3.1: Visualization of mappability in IGV. Example of the yeast genome: (a) - full genome, (b) - chrI, (c) - chrXII. On each subfigure three mappability tracks are shown corresponding to k-mer sizes: 30, 150 and 500 bp (from top to bottom of each figure).

bigger k-mers make the mappability values slightly higher, which can be observed in Figure 3.1b and 3.1c.

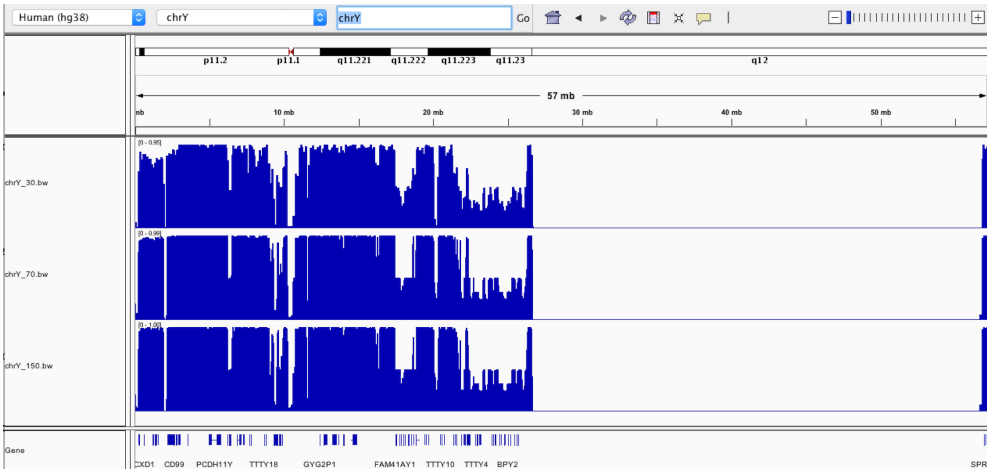


Figure 3.2: Visualization of mappability in IGV. Example of the human ChrY: three mappability tracks are shown corresponding to k-mer sizes: 30, 70 and 150 bp (from top to bottom). The completely unmappable region ($M_k(x)$=0) spans more than a half of the chromosome.

In contrast, the mappability track of the human chrY (see Figure 3.2) appears to be mostly "white", meaning that the mappability values are low on average. Overall, this results of the mappability visualization met our initial expectations. In the next section we will try to prove them correct.

3.2 Correctness of the mappability computation results

There are no publications on the mappability of the yeast or the last assembly of the human genome (hg38), and hence it was impossible to verify the computed results; therefore, we decided to look at some regions on the track and collate them with the data found on the genome databases recorded in the section 2.2.1.

First, we had a closer look at some regions on the chrY that appeared to be fully unmappable and compared the coordinates of those regions with the gaps coordinates taken from the UCSC table browser[1] (see Table 3.1). The gaps represent the unknown sequences of nucleotides (NNNs) of a different types, such as heterochromatin, contigs, telomeres and centromeres; the chrY is reach in such gaps, the biggest one spanning a region of 30 Mb (highlighted in lilac), visible in Figure 3.2 at the corresponding coordinates in the right part of the mappability track.

[1] `http://genome.ucsc.edu/cgi-bin/hgTables`

Table 3.1: Gaps on the human chrY.

chromStart	chromEnd	size (bp)	type
1949345	2132994	183649	contig
...	...	...	...
26673214	56673214	30000000	heterochromatin
56771509	56821509	50000	contig
57217415	57227415	10000	telomere

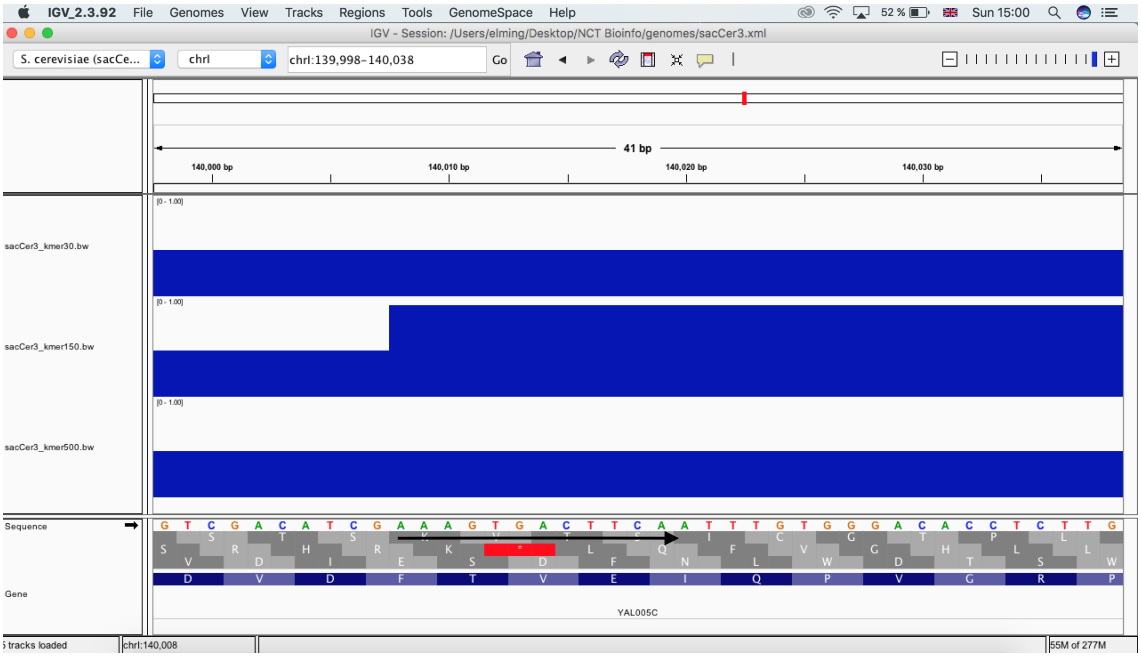
As next, we summed up the total gaps length and compared it with the length of the unmappable regions on the chrY (see section **??**) to see how much those regions differ in size. It resulted in a difference of roughly 30% - another evidence for the mappability importance and inability to retrieve the IS on the ∼80% of the chrY in case a viral vector inserted there.

Table 3.2: Gaps vs. unmappable regions on the chrY.

	Total gaps size	**Unmappable region k-mer 40**	**Unmappable region k-mer 100**	**Unmappable region k-mer 200**
chrY	30812367	47687448	46095768	45442177
% of the total length	53,84%	83,33%	80,55%	79,41%

To verify the results for the yeast genome two exact positions on the chrI and chrXII (see Figure 3.3) were chosen and the sequences of the respective k-mer lengths were aligned, starting from that exact position, onto the yeast genome itself using Megablast as described on page 13. The number of times this exact sequence was found within the yeast genome (frequency $F_k(x)$) up to the given number of mismatches was calculated (4%), and so the Blast-mappability was obtained (see Table 3.3); the GEM-mappability data was read in IGV by directing the cursor at the position of interest.

Both methods showed the same result proving the computation correct.



(a) Gene: SSA1 YAL005C

(b) Gene: ASP3-1 YLR155C

Figure 3.3: Positions at which the mappability values were compared with BLAST-results: (a) Chromosome I, position 140008, (b) Chromosome XII, position 469317.

Table 3.3: GEM-mappability vs. BLAST-mappability.

position	***k*-mer (bp)**	**GEM-mappability**	**Blast-mappability**
140008	30	0.5	0.5
140008	150	1.0	1.0
140008	500	0.5	0.5
469317	30	0.25	0.25
469317	150	0.25	0.25
469317	500	0.25	0.25

3.3 Uniquely mappable regions of the human genome.

In the next step of the Random Model One development the mappability of the whole human genome (hg38) was computed using three k-mer lengths $[40, 100, 200]$. These exact lengths were chosen according to the read length distribution in a typical IS retrieval pipeline. An example of the trimmed reads distribution was obtained from the paper by Calabria et al. and is shown in Figure 3.4 [5].

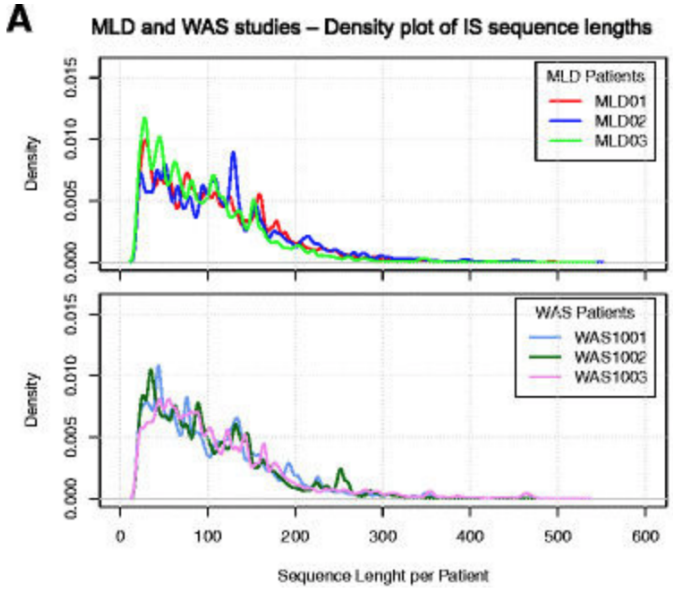


Figure 3.4: Distribution of trimmed IS reads obtained by Vector Integration Sites Parallel Analysis (VISPA) [5].

The lengths of the reads lay between ∼20 and ∼300 bp, and the k-mers are chosen accordingly to lay approx. in the middle of each 0.3 percentile.

The computation was performed overnight and took about 8 hours for each k-mer. The computation time could be reduced by specifying the approximation parameter t as explained in the section 2.2.1, but since the aim was to compute exact mappability this parameter was disabled.

For each k-mer and each chromosome the lengths of the uniquely mappable regions were computed and collected into the Tables 3.5, 3.6 and 3.7. They reveal, for instance, that the chromosomes differ strongly in their mappability level compared to each other: chrY, as expected, is the least mappable (∼20%) followed by the chromosomes 22 and 21. The average mappability of the human genome version hg38 accounts for ∼80% on average (for the exact numbers corresponding to the distinct k-mers see Table 3.4).

Table 3.4: Average mappability of the human genome (hg38).

k-mer (bp)	40	100	200
Average mappability	75.49%	83.57%	85.69%

Table 3.5: Uniquely mappable regions k-mer=40.

chr#	**size(bp)**	**Uniquely mappable regions (bp) k-mer=40**	**Percentage mappable regions k-mer40**
chr1	248956422	183895114	73,87%
chr2	242193529	200761632	82,89%
chr3	198295559	167230863	84,33%
chr4	190214555	158680130	83,42%
chr5	181538259	148972197	82,06%
chr6	170805979	138502962	81,09%
chr7	159345973	124765815	78,30%
chr8	145138636	119418047	82,28%
chr9	138394717	92552752	66,88%
chr10	133797422	109474593	81,82%
chr11	135086622	108833262	80,57%
chr12	133275309	108906942	81,72%
chr13	114364328	82811404	72,41%
chr14	107043718	72080449	67,34%
chr15	101991189	61651154	60,45%
chr16	90338345	59954439	66,37%
chr17	83257441	57158462	68,65%
chr18	80373285	64105197	79,76%
chr19	58617616	39940707	68,14%
chr20	64444167	51338706	79,66%
chr21	46709983	27789889	59,49%
chr22	50818468	26177198	51,51%
chrX	156040895	116930160	74,94%
chrY	57227415	9539967	16,67%

Table 3.6: Uniquely mappable regions k-mer=100.

chr#	**size(bp)**	**Uniquely mappable regions (bp) k-mer = 100**	**Percentage mappable regions k-mer = 100**
chr1	248956422	204268190	82,05%
chr2	242193529	219512880	90,64%
chr3	198295559	183361795	92,47%
chr4	190214555	173386256	91,15%
chr5	181538259	163126407	89,86%
chr6	170805979	152047698	89,02%
chr7	159345973	138971276	87,21%
chr8	145138636	130602405	89,98%
chr9	138394717	102161250	73,82%
chr10	133797422	121112010	90,52%
chr11	135086622	120151193	88,94%
chr12	133275309	121496919	91,16%
chr13	114364328	89972761	78,67%
chr14	107043718	79735905	74,49%
chr15	101991189	68492827	67,16%
chr16	90338345	67603560	74,83%
chr17	83257441	66057563	79,34%
chr18	80373285	69530997	86,51%
chr19	58617616	49095071	83,75%
chr20	64444167	56987346	88,43%
chr21	46709983	30474271	65,24%
chr22	50818468	29953487	58,94%
chrX	156040895	131578655	84,32%
chrY	57227415	11131647	19,45%

Table 3.7: Uniquely mappable regions k-mer=200.

chr#	**size(bp)**	**Uniquely mappable regions (bp) k-mer=200**	**Percentage mappable regions k-mer200**
chr1	248956422	209471710	84,14%
chr2	242193529	224441603	92,67%
chr3	198295559	187694908	94,65%
chr4	190214555	177558252	93,35%
chr5	181538259	166932212	91,95%
chr6	170805979	155653058	91,13%
chr7	159345973	142854303	89,65%
chr8	145138636	133378381	91,90%
chr9	138394717	104746422	75,69%
chr10	133797422	124027430	92,70%
chr11	135086622	123276333	91,26%
chr12	133275309	124680501	93,55%
chr13	114364328	91845743	80,31%
chr14	107043718	81649288	76,28%
chr15	101991189	70243737	68,87%
chr16	90338345	69266003	76,67%
chr17	83257441	68314473	82,05%
chr18	80373285	70939282	88,26%
chr19	58617616	51215221	87,37%
chr20	64444167	58423933	90,66%
chr21	46709983	31210181	66,82%
chr22	50818468	30933084	60,87%
chrX	156040895	135757250	87,00%
chrY	57227415	11785238	20,59%

3.4 Distribution of 10000 positions generated on chr1 with three random models

Finally, it is possible to visualize the distributions of the data points produced by RARE within one single chromosome and investigate the differencies between the random models. The chr1 was chosen for two reasons: it is the biggest one and has a centromere region in the central part, so the symmetry can be observed on the graphs, helping to follow the logic of the distribution functions (PDF and CDF).

Random position generation using the simplest random model RM^0 results in an, at a first glance, uniform distribution (see Figure 3.5). If the 10000 datapoints must be equally distributed among 100 bins, 100 positions in each bin is expected. In Figure 3.5 there is a slight fluctuation observed from this number, the least amount of data points accounting for 71 in the bin #58 and the largest number being 127 in the bin #57.

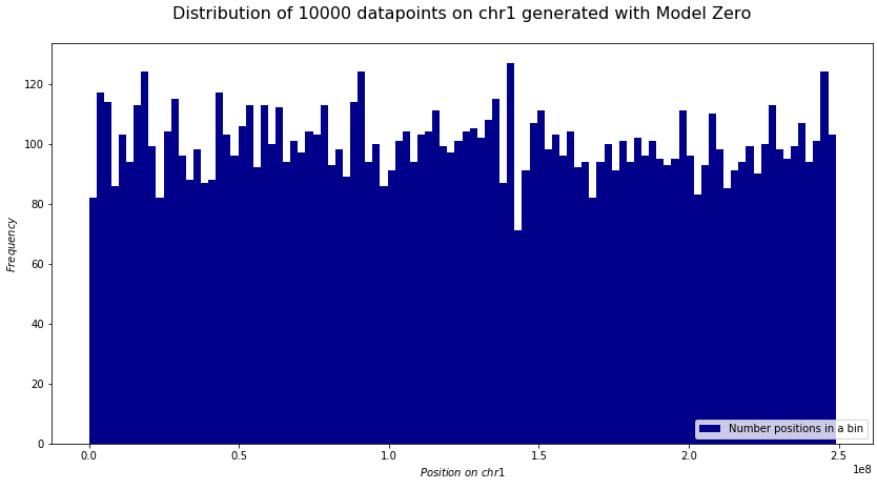


Figure 3.5: Distribution of 10000 positions on chr1 generated with RM^0.

For each of the bins in a histogram a two-sided binomial test was performed as described in the section 2.4.1. Below are the bins with an extreme number of points, for which the binomial test resulted in a P-value less than the significance level α of 5%. This corresponds to the expected type I error rate of 5%: incorrect rejection of a true null hypothesis. To minimize the type I error, the Bonferroni correction (see page 20) was applied and it resulted in all bins passing the binomial test since all P-values were greater than the corrected significance level of 0.0005.

bin#	Number of IS	P-value	Region on the chromosome (bp)
8	124	0.018	17426950 - 19916514
37	125	0.015	89624312 - 92113876
57	127	0.0089	139415596 -
58	71	0.0025	- 144394725
99	124	0.018	243977294 - 246466858

The conclusion, that can be drawn from this experiment, is that the distribution of the data points generated by RM^0 is truly random.

The second distribution was generated by RM^1 that constrained the position selection to only uniquely mappable regions of the respective chromosome. This fact is visible in the distribution pattern on Figure 3.6: there are bins in the middle and on both sides of the graph with extremely low frequency.

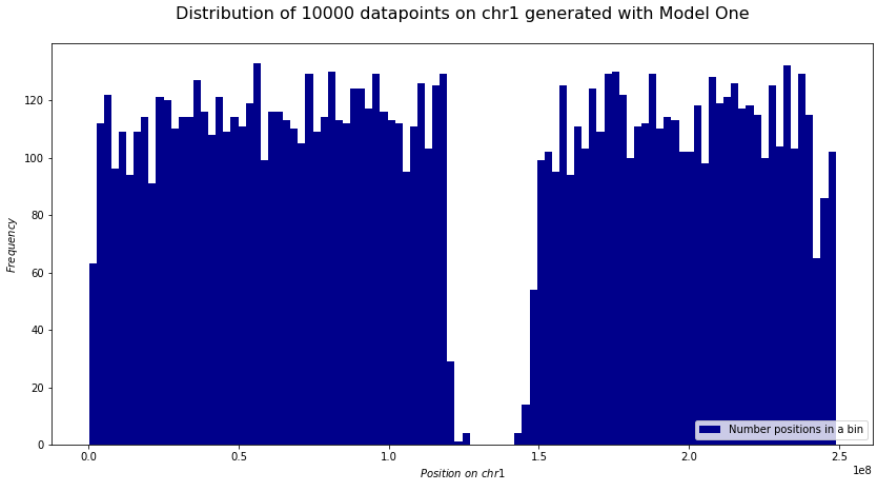


Figure 3.6: Distribution of 10000 positions on chr1 generated with RM^1.

The expected amount of data points in each bin under the uniform distribution assumption is now greater than 100, since the "real" number of available bins is lower (74), and accounts for 135 points (see section 2.4.1 for explanation). In total, there are 20 bins that don't pass the binomial test after the Bonferroni correction.

bin#	Number of IS	P-value	Region on the chromosome (bp)
1	63	5.89e-12	0 - 2489564
6	93	1.30e-04	12447821 - 14937385
9	92	9.46e-05	19916514 - 22406078
43	95	3.17e-04	104561697 - 107051261
49	29	1.77e-28	119499083 - 121988647
50	1	2.63e-57	121988647 -

```
51      4               3.15e-52
52      0               1.81e-59
53      0               1.81e-59
54      0               1.81e-59
55      0               1.81e-59
56      0               1.81e-59
57      0               1.81e-59
58      4               3.15e-52
59      14              1.53e-40
60      54              2.84e-15        - 149373853
63      95              3.17e-04        154352982 - 156842546
65      93              1.36e-04        159332110 - 161821674
98      62              2.35e-12        241487729 -
99      85              4.21e-06        - 246466858
```

A couple of values at the beginning and at the end of the graph as well as a large number of values in the middle part don't obey the uniform distribution rule, the latter representing a centromere that spans a region between 125184587 and 143184587 bp (bins# 51-58).

Finally, the last distribution generated by RM^2 has a different shape compared to the previous two distributions (see Figure 3.7). This random model brings another constrain into the position generation: each data point is matched to the restriction site of the MseI enzyme. There are certainly more cleavage cites distributed around the centromeric regions than at the beginning of the chromosome.

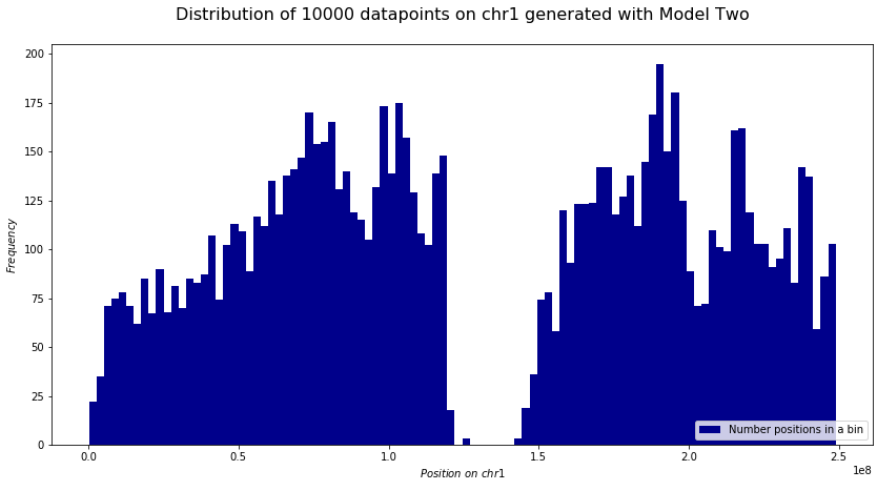


Figure 3.7: Distribution of 10000 positions on chr1 generated with RM^2 matched to the MseI cleavage sites.

As expected, even more bins (45) fail the binomial test compared to the previous distribution, reflecting the non-random character of the cleavage sites allocation on chr1. In fact, only half of the frequencies follow the uniform distribution corresponding to the bins# 23-48, 66-76 and a couple of other separated bins.

bin#	Number of IS	P-value	Region on the chromosome (bp)
1	22	1.45e-33	0 -
2	31	5.19e-27	
3	71	1.51e-09	
4	75	2.05e-08	
5	79	1.87e-07	
6	72	3.31e-09	
7	62	2.35e-12	
8	82	9.31e-07	
9	68	2.11e-10	
10	90	4.50e-05	
11	70	8.57e-10	
12	82	9.31e-07	
13	68	2.11e-10	
14	86	7.73e-06	
15	84	2.74e-06	
16	84	2.74e-06	- 39833028
18	80	3.70e-07	42322592 -
19	96	4.43e-04	- 47301720
22	88	1.74e-05	52280849 - 54770413
49	19	7.39e-36	119499083 -

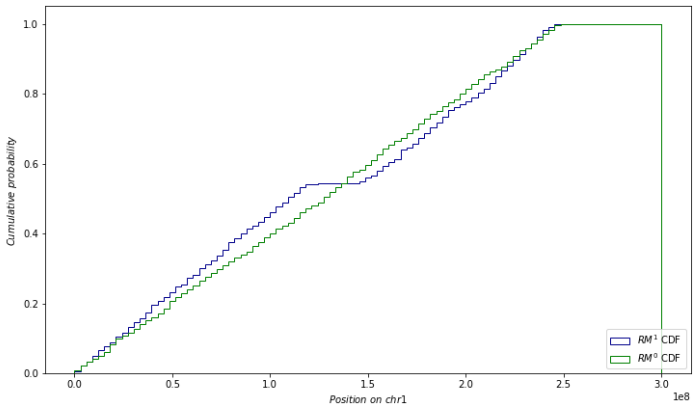
50	0	1.81e-59	
51	3	7.56e-54	
52	0	1.81e-59	
53	0	1.81e-59	
54	0	1.81e-59	
55	0	1.81e-59	
56	0	1.81e-59	
57	0	1.81e-59	
58	3	7.56e-54	
59	18	7.74e-37	
60	37	1.68e-23	
61	73	5.68e-09	
62	79	1.87e-07	
63	58	9.05e-14	- 156842546
65	93	1.36e-04	159332110 - 161821674
77	196	7.32e-07	189206881 - 191696445
79	180	1.92e-04	194186009 - 196675573
81	89	2.58e-05	199165138 -
82	71	1.51e-09	
83	72	3.31e-09	- 206633830
92	91	6.54e-05	226550344 -
93	95	3.17e-04	231529472
95	82	9.31e-07	234019037 - 236508601
98	61	1.22e-12	241487729 -
99	85	4.21e-06	- 246466858

It is interesting to note that the bin#43 has failed the binomial test in the RM^1 distribution, but passed it in the RM^2, meaning that the region #43 is poorly mappable but has a lot of 'TTAA' sequences in the non-repetitive parts. The bins #77 and #79 also appear to be very reach in MseI restriction sites, making the number of IS generated in the bin unexpectedly high.

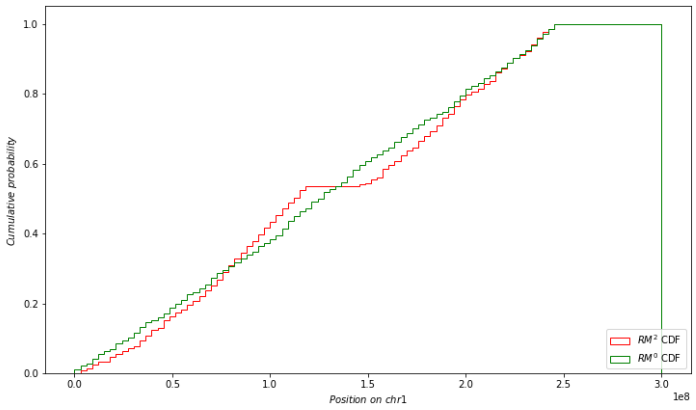
Another method to compare the distributions discussed above is to plot their CDFs and apply the Kolmogorov-Smirnov statistical test.

In Figure 3.8 the distribution of 1000 positions generated with three random models was converted into the cumulative probability histogram and plotted pairwise for the sake of a better visualization.

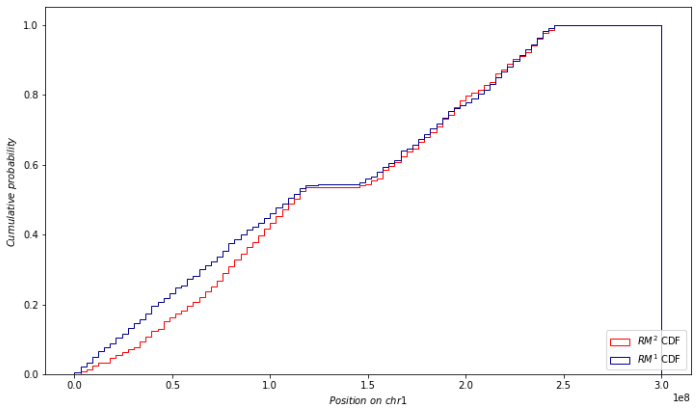
In Figure 3.8a at the beginning of the graph it is worth to draw attention on the steeper character of the RM^1 curve compared to RM^0; it demonstrates that the bigger portion of data points accumulates in the first half of the chromosome compared to the simple random model CDF that grows uniformly over all the coordinates. In



(a) RM^0 vs. RM^1



(b) RM^0 vs. RM^2



(c) RM^1 vs. RM^2

Figure 3.8: Pairwise comparison of CDFs.
K-S test results: (a) - D-statistic=0.08399, P-value=0.00159, (b) - D-statistic=0.06800, P-value=0.01864), (c) - D-statistic=0.08200, P-value=0.00223.

the middle of the chromosome the RM^1 curve stops growing, since no positions were generated within the unmappable centromeric part; after, it starts recovering again with a slightly steeper slope than RM^0.

Figure 3.8b represents the CDFs drawn from the simple random and enzyme bias models. Here it is noticeable that the the first quarter of the chromosome accounts for less enzyme cleavage sites than expected in the truly random distribution (the red curve is below the green one). In the second quarter it recovers and reaches the same amount of datapoints as the RM^1 curve in 3.8a. In the third quarter RM^2 is again steeper than RM^0, but the red curve is below the green since no data points were generated in and around the centromere. The two curves then combine in the last quarter of the chromosome coordinates.

Finally, the last graph represents a comparison between Random Models One and Two. The biggest difference is observed in the first half of the chromosome where the steep character of the blue curve is confronted with a drooped growth of the red one, again demonstrating a depletion in the cutting sites for the MseI enzyme.

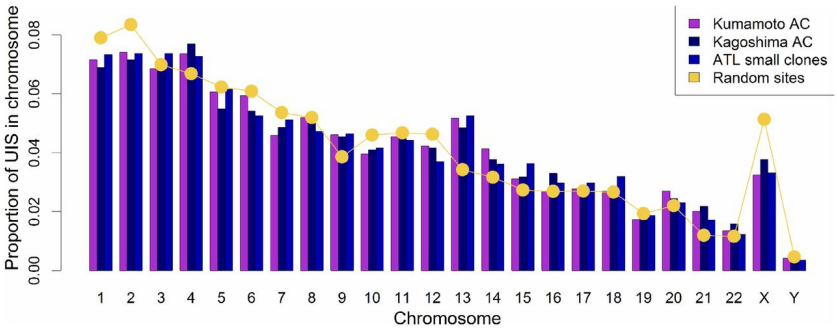
All three K-S tests are showing significant differences among the distributions (P-value less than 0.05) proving the null hypothesis, that the samples of data points come from the same population, to be wrong.

3.5 Analysis of HTLV-1 integration landscape

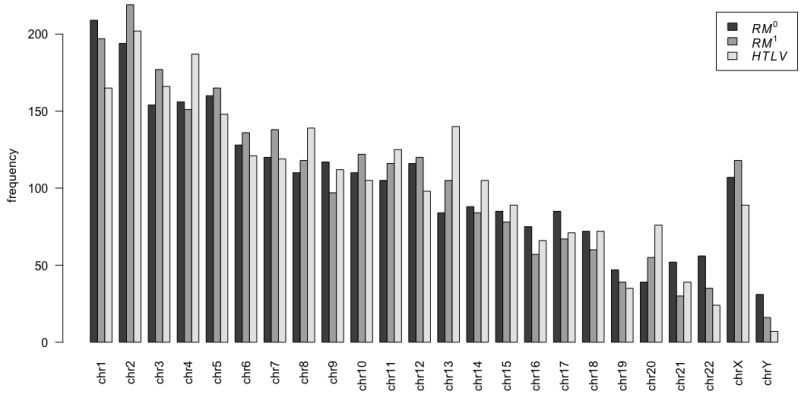
In the last section of this chapter we will compare the result of the IS analysis, described in the paper by Cook et al. [7], with the one produced using the RARE framework. In the study of the HTLV-1 integration landscape the authors show the distribution of the experimental data set against the random one without providing details on the kind of the random simulator they used. They state that HTLV-1 is integrating into the host genome in a random manner with a statistically significant preference towards chromosomes 13, 14, 15, and 21.

In Figure 3.9b the HTLV-1 IS distribution is plotted against two models developed in this Bachelor-project RM^0 and RM^1, and in Figure 3.9a the original graph from the paper is included. There was no reason to use the RM^2 since the HTLV-1 IS retrieval was based on DNA shearing with no restriction enzymes involved.

A pairwise comparison of the data for each chromosome provided an evidence that RM^1 repeats exactly the pattern showed in the paper and in most of the cases better than RM^0. The first conclusion, that can be drawn from this observation,



(a) HTLV-1 IS distribution by Cook et al. [7]



(b) HTLV-1 IS distribution compared to RM^1 and RM^2

Figure 3.9: Juxtaposition of the results from the paper by Cook et al. [7] with the data sets generated with RARE.

is that something similar to the Random Model One was used by the authors to generate the random data set. In fact, in an earlier study by the same research group it was specified that the random IS were backaligned to the human genome using a Galaxy pipeline to eliminate any potential bias due to alignment limitations [21].

As next, we analyzed contingency tables including 2500 IS, from which the distributions in Figure 3.9 were drawn. Again, the binomial test was used to prove the null hypothesis that there is no significant difference between the expected number of IS according to the RM^1 and the experimentally obtained number of insertions in each chromosome. The test showed that it was only possible to reject the null hypothesis for two chromosomes (chr13 and chrX) and one test showed a borderline result (chr14), meaning HTLV-1 preferentially chooses chr13 and chr14 for integration and avoids chrX compared to the random data set.

```
chr#   P-value
chr13  1.03969758055e-07
chr14  0.00213790983128
chrX   0.000319062135551
```

The results described above intersect with the results in the paper regarding chr13 and chr14. However, for chromosome 15 and 21 no significant difference between experimental and random data sets could be detected.

4 Conclusion

The purpose of this Bachelor project was to develop a flexible and biologically correct framework for random data set generation to use in the IS and insertional mutagenesis analysis. The goal was achieved by splitting the framework into three models: the simplest random region generator (RM^0), the mappability model (RM^1) and the enzyme bias model (RM^2). Modern technologies such as LAM-PCR and IS retrieval pipelines (GENE-IS [1], VISPA[5] etc.) imply the loss of information during the analysis. On the one hand, LAM-PCR is unable to detect viral vector insertions far away from the restriction enzyme cleavage sites due to disfavouring of long fragments in the amplification process. On the other hand, realignment of the amplified reads sometimes results in multiple coordinates, and those reads have to be discarded. The IS detection is therefore constrained to uniquely mappable parts of the genome.

While waiting for the new better technologies able to detect IS without producing a bias to evolve, it appears to be reasonable now to adjust the random data set and make it imitate the experimental data set associated restrictions. In this thesis the mappability of the human genome (hg38) was computed using three different k-mers: $\sim$20% of the genome is unmappable (see Table 3.4). The Random Models One and Two cope with this fact by limiting position generation to uniquely mappable regions only.

In the section 3.4 we discussed the distribution of 10000 positions generated with each of the three random models on the chr1. We found out that the character of the distribution was changing from completely uniform in case of the RM^0 to semi-uniform in case of the RM^2, which was proven by the binomial test. Additionally, K-S test was applied to look at the cumulative distribution functions of the random models and at the distances between them. This, again, revealed that the models follow significantly different distribution patterns. In spite of the fact that the Models One and Two are not perfectly random in terms of computer generated sequences, they appear to be more correct in terms of biological analysis of the viral vector IS since they imitate the limitations in the experimental data retrieval methods.

Finally, we compared the HTLV-1 integration landscape results obtained by Cook et al. with our own plot and found their random model more similar to our RM^1 than to RM^0. This suggests that the authors most probably considered the mappability in the random data set generation. As next we proved, whether using our framework we will be able to draw the same conclusions about preferential insertion of HTLV-1 into the chromosomes 13, 14, 15, and 21. This was partly

possible: chromosome 13 and 14 were proven to be preferentially chosen by the virus for integration, but for chromosomes 15 and 21 we couldn't detect any significant difference. Overall, the framework appeared to be suitable for such an analysis, providing similar results to the published ones [7].

The RARE framework has a potential to be further improved in terms of development of a user-friendly pipeline for a random data set generation suitable for any demand. Furthermore, it would be interesting to test it with some historical data sets and look for the prospective new insights into the older studies. New experimental data can be also analyzed using the framework and gain flexibility in the random control data set generation.

Acronyms

$\boldsymbol{RM^0}$	Random Model Zero: Simple model
$\boldsymbol{RM^1}$	Random Model One: Mappability
$\boldsymbol{RM^2}$	Random Model Two: Enzyme bias
1D	one-dimensional
2D	two-dimensional
AAV	Adeno-associated virus
ASLV	Avian sarcoma leukosis virus
BMI1	B cell-specific Moloney murine leukaemia virus integration site 1 proto-ocogene
CDF	Cumulative distribution function
chrY	human chromosome Y (assembly hg38)
CREs	*cis*-regulatory elements
hg38	human genome assembly Dec. 2013
HIV	Human immunodeficiency virus
HTLV-1	Human T-lymphotropic virus 1
IGV	Integrative Genomics Viewer
IS	integration site
K-S test	Kolmogorov-Smirnov test
L1	long interspersed element-1
LAM-PCR	linear amplification-mediated PCR
LC	linker cassette
LMO2	proto-oncogene LIM domain only 2
LTR	long terminal repeat
MLV	Murine leukemia virus
NGS	next generation sequencing
nrLAM-PCR	non-restrictive LAM-PCR
PDF	Probability distribution function
RARE	framework for random region generation
Re-free LAM-PCR	restriction enzyme-free LAM-PCR

ROC	receiver operator characteristic
SB	Sleeping beauty
SFV	Simian foamy virus
TU	transcription unit
UCSC	University of California, Santa Cruz
VISPA	Vector Integration Sites Parallel Analysis
X-SCID	X-linked severe combined immunodeficiency

Bibliography

[1] Saira Afzal et al. “GENE-IS: Time-Efficient and Accurate Analysis of Viral Integration Events in Large-Scale Gene Therapy Data”. In: *Molecular Therapy - Nucleic Acids* 6 (2017), pp. 133–139.

[2] Alessandro Aiuti et al. “Multilineage hematopoietic reconstitution without clonal selection in ADA-SCID patients treated with stem cell gene therapy”. In: *The Journal of Clinical Investigation* 117.8 (Aug. 2007), pp. 2233–2240.

[3] K. Akagi et al. “RTCGD: retroviral tagged cancer gene database”. In: *Nucleic Acids Research* 32.90001 (Jan. 2004), pp. 523D–527.

[4] Charles Berry et al. “Selection of Target Sites for Mobile DNA Integration in the Human Genome”. In: *PLOS Computational Biology* 2.11 (Nov. 2006), pp. 1–13.

[5] Andrea Calabria et al. “VISPA: a computational pipeline for the identification and analysis of genomic vector integration sites”. In: *Genome Medicine* 6.9 (2014), p. 67.

[6] Nathalie Cartier et al. “Hematopoietic Stem Cell Gene Therapy with a Lentiviral Vector in X-Linked Adrenoleukodystrophy”. In: *Science* 326.5954 (2009), pp. 818–823. eprint: `http://science.sciencemag.org/content/326/5954/818.full.pdf`.

[7] Lucy B. Cook et al. “The role of HTLV-1 clonality, proviral structure, and genomic integration site in adult T-cell leukemia/lymphoma”. In: *Blood* 123.25 (2015), pp. 3925–3932.

[8] Thomas Derrien et al. “Fast Computation and Applications of Genome Mappability”. In: *PLOS ONE* 7.1 (Jan. 2012), pp. 1–16.

[9] Richard Gabriel et al. “Comprehensive genomic access to vector integration in clinical gene therapy”. In: *Nat Med* 15.12 (Dec. 2009), pp. 1431–1436.

[10] Richard Gabriel et al. “Linear Amplification Mediated PCR - Localization of Genetic Elements and Characterization of Unknown Flanking DNA”. In: *Journal of Visualized Experiments* 88 (2014), e51543–e51543.

[11] H. Bobby Gaspar et al. “Hematopoietic Stem Cell Gene Therapy for Adenosine Deaminase–Deficient Severe Combined Immunodeficiency Leads to Long-Term

Immunological Recovery and Metabolic Correction". In: *Science Translational Medicine* 3.97 (2011), 97ra80–97ra80.

[12] Salima Hacein-Bey-Abina et al. "A Serious Adverse Event after Successful Gene Therapy for X-Linked Severe Combined Immunodeficiency". In: *New England Journal of Medicine* 348.3 (2003), pp. 255–256.

[13] Salima Hacein-Bey-Abina et al. "Insertional oncogenesis in 4 patients after retrovirus-mediated gene therapy of SCID-X1." In: *The Journal of clinical investigation* 118.9 (2008), pp. 3132–42.

[14] Steven J Howe et al. "Insertional mutagenesis combined with acquired somatic mutations causes leukemogenesis following gene therapy of SCID-X1 patients." In: *The Journal of clinical investigation* 118.9 (2008), pp. 3143–50.

[15] Mark A. Jobling and Chris Tyler-Smith. "The human Y chromosome: an evolutionary marker comes of age". In: *Nature Reviews Genetics* 4.8 (2003), pp. 598–612.

[16] Donald B. Kohn, Michel Sadelain, and Joseph C. Glorioso. "Occurrence of leukaemia following gene therapy of X-linked SCID". In: *Nat Rev Cancer* 3.7 (July 2003), pp. 477–488.

[17] Neeltje A. Kootstra and Inder M. Verma. "Gene Therapy with viral vectors". In: Vol.43 (April 2003), pp. 413–439.

[18] Zhixiong Li et al. "Murine Leukemia Induced by Retroviral Gene Marking". In: *Science* 296.5567 (2002).

[19] Dan Lin et al. "Re-Annotation of Protein-Coding Genes in the Genome of Saccharomyces cerevisiae Based on Support Vector Machines". In: *PLoS ONE* 8.7 (2013), e64477.

[20] Santiago Marco-Sola et al. "The GEM mapper: fast, accurate and versatile alignment by filtration". In: *Nat Meth* 9.12 (2012), pp. 1185–1188.

[21] Anat Melamed et al. "Genome-wide Determinants of Proviral Targeting, Clonal Abundance and Expression in Natural HTLV-1 Infection". In: *PLoS Pathogens* 9.3 (2013). Ed. by Michael Emerman, e1003271.

[22] Ute Modlich et al. "Leukemias following retroviral transfer of multidrug resistance 1 (MDR1) are driven by combinatorial insertional mutagenesis". In: *Blood* 105.11 (2005), pp. 4235–4246.

[23] Luigi Naldini. "Gene therapy returns to centre stage". In: *Nature* 526.7573 (2015), pp. 351–360.

[24] Marion G Ott et al. "Correction of X-linked chronic granulomatous disease by gene therapy, augmented by insertional activation of MDS1-EVI1, PRDM16 or SETBP1". In: *Nature Medicine* 12.4 (2006), pp. 401–409.

[25] William H. Press et al. *Numerical Recipes in C: the art of scientific computing*. Second Edition. Cambridge University Press, 1992.

[26] Isabelle Rivière, Cynthia E. Dunbar, and Michel Sadelain. "Hematopoietic stem cell engineering at a crossroads". In: *Blood* 119.5 (Feb. 2012), p. 1107.

[27] Michel Sadelain, Eirini P. Papapetrou, and Frederic D. Bushman. "Safe harbours for the integration of new DNA in the human genome". In: *Nature Reviews Cancer* 12.1 (2011), pp. 51–58.

[28] Manfred Schmidt et al. "High-resolution insertion-site analysis by linear amplification-mediated PCR (LAM-PCR)". In: *Nat Meth* 4.12 (Dec. 2007), pp. 1051–1057.

[29] D.S. Starnes, D. Yates, and D.S. Moore. *The Practice of Statistics*. W. H. Freeman, 2010.

[30] Clare E. Thomas, Anja Ehrhardt, and Mark A. Kay. "Progress and problems with the use of viral vectors for gene therapy". In: *Nat Rev Genet* 4.5 (May 2003), pp. 346–358.

[31] Chuanfeng Wu et al. "High efficiency restriction enzyme-free linear amplification-mediated polymerase chain reaction approach for tracking lentiviral integration sites does not abrogate retrieval bias." In: *Human gene therapy* 24.1 (2013), pp. 38–47.

Supplemental

Random model Zero: Simple Model

Python script[1] for the random dataset generation with RM^0

```
#parser setup
def parseArgs():
    import sys
    import argparse
    parser = argparse.ArgumentParser()
    parser.prog = 'progName.py'
    parser.description = 'You can provide the program with three
        parameters through the terminal'
    parser.add_argument("-n", type=int, help='Number of integration
        regions generated')
    parser.add_argument("-r", type=int, help='Range-value: defines an
        interval where the IR is located')
    parser.add_argument("-d", type=int, help='Delta-value: expands the
        range with the value provided by user')
    #add new input argument
    parser.add_argument("-i", help='Input-file .txt with names of
        chromosomes and their lengths in bp organized into two columns and
        separated by \t')
    namespace = parser.parse_args((sys.argv[1:]))
    return namespace
#a function that proves whether the input file contains two columns
def checkFormatting():
    check=True
    #go through all of the lines and check whether exactly two columns are
         present
    for line in dimension_file:
        result=line.split()
        if len(result)==2:
            continue
        else:
            print 'Checking format: the text file has to contain two
                columns'
            check=False
            break
    if check == True:
        print "Checking format: the file is well formatted"
    return check
```

[1]Saved under `getRegionW2.py` `https://github.com/ElviraMingazova/RARE/tree/master/scripts`

```
#a function that proves whether the first column contains a string "chrXY"
     where X and Y are digits
def checkColumn1():
    check = True
    column1=[]
    for line in dimension_file:
        if line.split()[0].strip() not in column1:
            column1.append(line.split()[0].strip())
        else:
            print "Checking the first column: one of the chromosome names
                is duplicated"
            check = False
            break
    if check == True:
        print "Checking the first column: the names of chromosomes are
            correct"
    return check

#a function that proves whether the second column contains a positive
    integer
def checkColumn2():
    check = True
    column2=[]
    for line in dimension_file:
        try:
            #extract the second column to a list and try to convert each
                element into an integer
            column2.append(int(line.split()[1].strip()))
        #catch a ValueError: invalid literal for int()
        except ValueError:
            check = False
            print "Checking the second column: could not convert {} to an
                integer".format(line.split()[1].strip())
    if check == True:
        for length in column2:
            if length<0:
                print "Checking the second column: you provided a negative
                    chromosome length: {}".format(length)
                check = False
    if check == True:
        print "Checking the second column: the lengths of chromosomes are
            correct"
    return check
```

```
#a function which will process the given text file and return {"chr name":
     chr length} pairs inside of a dictionary
def getDimensionDict(filename):
    d={}
    with open(filename) as dimension_file:
        for line in dimension_file:
            key, value = line.split()
            d[key.strip()]=int(value.strip())
        dimension_file.close()
    return d
#the following line will protect the code frome executing if I will import
     the file as a module
if __name__ == "__main__":
    #check the input file
    namespace = parseArgs()
    f = open(namespace.i,"r")
    dimension_file=f.readlines()
    check=checkFormatting()
    if check==True:
        check=checkColumn1()
        if check==True:
            check=checkColumn2()
            if check == True:
                dimension_dict = getDimensionDict(namespace.i)
                #loop on n random IR
                import random
                import numpy
                count = 1
                print "chr#", '\t', 'Start', '\t\t', 'End', '\t\trnd#'
                #weighted chromosome choice
                weights = {k:v/sum(dimension_dict.values()) for k,v in
                    dimension_dict.iteritems()}
                chroms, probs = zip(*weights.iteritems())
                for i in range(namespace.n):
                    #select a chromosome
                    chrom = numpy.random.choice(chroms, 1, p=probs)

                    #select a site on that chromosome
                    start = random.randint(1,dimension_dict[chrom[0]])

                    #select a random region
                    end = start + random.randint(0,namespace.r) + namespace.
                        d
                    if len(str(chrom[0]))==1:
```

```
                    print chrom[0],'',"\t",start,"\t",end,"\trnd{}".
                        format(count)
                else:
                    print chrom[0],"\t",start,"\t",end,"\trnd{}".format(
                        count)
                count +=1
f.close()
```

Random model One: Mappability

Example python script[1]

```
#total lengths of the chromosomes
dimension_dict={'chr13': 114364328, 'chr12': 133275309, 'chr11':
    135086622, 'chr10': 133797422, 'chr17': 83257441, 'chr16': 90338345, '
    chr15': 101991189, 'chr14': 107043718, 'chr19': 58617616, 'chr18':
    80373285, 'chr24': 57227415, 'chr22': 50818468, 'chr23': 156040895, '
    chr20': 64444167, 'chr21': 46709983, 'chr7': 159345973, 'chr6':
    170805979, 'chr5': 181538259, 'chr4': 190214555, 'chr3': 198295559, '
    chr2': 242193529, 'chr1': 248956422, 'chr9': 138394717, 'chr8':
    145138636}
#dictionaries containing the factors to calculate the number of positions
    to be generated on the respective chromosome
kmer40={'chr1':0.0788750,'chr2':0.086109,'chr3':0.071728,'chr4':0.068060,'
    chr5':0.063896,'chr6':0.059406,'chr7':0.053514,'chr8':0.051220,'chr9
    ':0.039697,'chr10':0.046955,'chr11':0.046680,'chr12':0.046712,'chr13
    ':0.035519,'chr14':0.030916,'chr15':0.026443,'chr16':0.025715,'chr17
    ':0.024516,'chr18':0.027496,'chr19':0.017131,'chr20':0.022020,'chr21
    ':0.011919,'chr22':0.011228,'chrX':0.050153,'chrY':0.004092}
kmer100={'chr1':0.079149,'chr2':0.085056,'chr3':0.071048,'chr4':0.067183,'
    chr5':0.063207,'chr6':0.058915,'chr7':0.053848,'chr8':0.050605,'chr9
    ':0.039585,'chr10':0.046928,'chr11':0.046556,'chr12':0.047077,'chr13
    ':0.034862,'chr14':0.030896,'chr15':0.026539,'chr16':0.026195,'chr17
    ':0.025596,'chr18':0.026942,'chr19':0.019023,'chr20':0.022081,'chr21
    ':0.011808,'chr22':0.011606,'chrX':0.050983,'chrY':0.004312}
kmer200={'chr1':0.079156,'chr2':0.084813,'chr3':0.070927,'chr4':0.067097,'
    chr5':0.063081,'chr6':0.058819,'chr7':0.053983,'chr8':0.050402,'chr9
    ':0.039582,'chr10':0.046868,'chr11':0.046584,'chr12':0.047115,'chr13
    ':0.034707,'chr14':0.030854,'chr15':0.026544,'chr16':0.026175,'chr17
    ':0.025815,'chr18':0.026807,'chr19':0.019354,'chr20':0.022078,'chr21
    ':0.011794,'chr22':0.011689,'chrX':0.051301,'chrY':0.004455}
import numpy as np
from __future__ import print_function
def getMappableIR(inputf,chr_name,kmer,mapname,N,outputf):
    """Generate N*kmer[chr_name] positions on a respective chromosome
       using the map of choice, where N is the total number of positions
       to be generated on the genome"""
    with open(inputf,"r"):
        chr_arr = np.genfromtxt(inputf,dtype=None)
    counter = 1
    while counter<=round(N*kmer[chr_name]):
```

[1]Saved under `mappable_randomIR.py` https://github.com/ElviraMingazova/RARE/tree/master/scripts

```
        rand_num = np.random.randint(1,dimension_dict[chr_name])
        matched_index = np.searchsorted(chr_arr["f0"],rand_num)-1
        if chr_arr["f1"][matched_index] == True:
            #append to the file that has been previously created with
                headers
            with open(outputf, 'a') as f:
                print (chr_name,"\t",rand_num,"\t",rand_num+1,"\t", "
                    unmatched","\t",mapname,"\t", "hg38", file=f)
                counter+=1
        else:
            continue

#first write a new file with tables header (example kmer=40, N=100000):
with open('results/map40_IS100000.txt', 'w') as f:
    print("Chromosome\tStart\tEnd\tMatch\tRnd Model\tAssembly", file=f)

listofchr = ["chr1","chr2","chr3","chr4","chr5","chr6","chr7","chr8","chr9
    ","chr10","chr11","chr12","chr13","chr14","chr15","chr16","chr17","
    chr18","chr19","chr20","chr21","chr22","chrX","chrY"]

#for each chromosome compute the corresponding number of IR and append to
    the file
for chrom in listofchr:
    getMappableIR('textfiles/map40/{0}_40_mappable.txt'.format(chrom),
        chrom, kmer40, "map_40", 100000, 'results/map40_IS100000.txt')
```

Random model Two: Enzyme Bias

Modified python function for the random dataset generation matched to an enzyme cleavage site

```
def getEnzMatchedIR(inputenzf,inputmappf,chr_name,kmer,N,mapname,enzname,
    outputf):
    enz = np.sort(np.genfromtxt(inputenzf, dtype=int))
    chr_arr = np.genfromtxt(inputmappf,dtype=None)
    counter = 1
    while counter<=round(N*kmer[chr_name]):
        rand_num = np.random.choice(enz)
        matched_index = np.searchsorted(chr_arr["f0"],rand_num)-1
        if chr_arr["f1"][matched_index] == True:
            with open(outputf, 'a') as f:
                print (chr_name,"\t",rand_num,"\t",rand_num+1,"\t", enzname
                    ,"\t",mapname,"\t", "hg38", file=f)
                counter+=1
        else:
            continue
#create the file with the headers
with open('MseI_map40_IS100000.txt', 'w') as f:
    print("Chromosome\tStart\tEnd\tMatch\tRnd Model\tAssembly", file=f)
#how to use
getEnzMatchedIR("MseI_chr1_pos.txt", "chr1_40_mappableshort.txt","chr1",
    kmer40,100000,"map_40","MseI",'MseI_map40_IS100000.txt')
```