Bioinformatics Methods

Forward Epigenetics using a CRISPR-based Screening Method

Anna Köferle

A thesis presented for the degree of
Doctor of Philosophy

University College London
London
October 2016

Contents

Chapter 1

Degenerate gRNA libraries

1.1 Finding all gRNA targets in the repeat-masked human genome

The genomic target sites of gRNAs designed for use with the *S.pyogenes* CRISPR/-Cas system are of the general form $GN_{20}GG$. The presence of the G at the start of the motif is required for initiation of transcription from the U6 promoter, present on many gRNA expression vectors. This is followed by 19 random bases (N) that determine the gRNA target site - these bases comprise the gRNA protospacer. The protospacer sequence is followed by the protospacer-adjacent motif (PAM), which is not part of the gRNA sequence but is present in the genomic target sequence just 3' of the gRNA target site. For the *S.pyogenes* CRISPR/Cas system, the PAM has to be comprised of the bases NGG.

A gRNA library is a pooled collection of gRNAs. A random library has a complexity of 4^{19}, which corresponds to 10^{12} different sequences. The aim here is to reduce this complexity and maximise binding to the genome. To this end, I first identified the number of occurences of the sequences GN20GG in the human genome. The Bioconductor (Bioconductor version 2.14) package BSgenome [15] was used to identify all sequences matching this pattern in the repeat-masked human genome in R (version 3.1.1). An R script named `find_all_gRNAs.R` contains the code and outputs a file containing chromosome coordinates and strand information for all hits, with the last column containing the pattern identifier. The script looks for the $GN_{20}GG$ by default (and outputs a file named `GN20GG_masked_allregions.txt`), unless another pattern is supplied by the user as follows:

```
./find_all_gRNAs.R -p [pattern, e.g. "GTACN"], -n [pattern
   -name, e.g. "my-pattern"], -o [outputfilename, e.g. "
   All_GTACN_hg19_RM.txt"]
```

The regions mapping to chromosomes 1-22 and the sex chromosomes were extracted from the outputfile `GN20GG_masked_allregions.txt` as follows:

```
grep -v 'random' GN20GG_masked_allregions.txt | grep -v '
```

```
   hap' | grep -v
'chrUn' | grep -v chrM > GN20GG_masked_autoXY.txt;

#remove the last column
awk '{print $1 "\t" $2 "\t" $3 "\t" $4}'
   GN20GG_masked_autoXY.txt >
GN20GG_masked_autoXY.bed;
```

1.2 Identifying gRNAs that overlap with known promoters

In order to generate an annotation file containing promoter regions, the annotation file for known transcripts (human GRCh37-70) was downloaded from Ensembl (version70) and parsed through a script supplied by Gareth Wilson. This script can be found here: `https://github.com/regmgw1/regmgw1_scripts/blob/master/ensembl_scripts/transcript2promoter.pl`). This script extracts coordinates -1000 and +500 bp from the start of the transcript. From this file non-overlapping promoter sequences were derived by strand-specific merging using the Bedtools suite (v2.17.0) [16].

```
#strand-specific merging of promoter annotation file
mergeBed -s -i promoters.gff | awk '{print "chr"$1 "\t" $2
    "\t" $3 "\t" $4}' > promoters_merged.bed

#use Bedtools to find regions that overlap promoter
# regions with minimum of 1 bp
intersectBed -a GN20GG_masked_autoXY.bed -b
   yourpath2annotation_files/
human_GRCh37_70/promoters_merged.bed -wa >
GN20GG_masked_autoXY_promoters_merged;
```

Then, FASTA coordinates were retrieved using twoBitToFa [8], which is available from the UCSC website (`http://hgdownload.cse.ucsc.edu/admin/exe/`).

```
#Separate according to whether pattern is on plus or minus
    strand
grep '+' GN20GG_masked_autoXY_promoters_merged >
GN20GG_masked_autoXY_promoters_merged_PLUS;
grep '-' GN20GG_masked_autoXY_promoters_merged >
GN20GG_masked_autoXY_promoters_merged_MINUS

#make into a gff file
awk '{print $1 ":" ($2 - 1) "-" $3}'
   GN20GG_masked_autoXY_promoters_PLUS  >
GN20GG_masked_autoXY_promoters_PLUS.gff;

twoBitToFa yourpath2/human/GRCh37/hg19.2bit
GN20GG_masked_autoXY_promoters_merged_PLUS.fa
-seqList=GN20GG_masked_autoXY_promoters_merged_PLUS.gff

#repeat for file containing hits on the minus strand
```

The Python script `reverse_complement_fasta.py` was used to reverse-complement the FASTA sequences on the minus strand [4]. The script was invoked as follows:

```
python reverse_complement_fasta.py
   GN20GG_masked_autoXY_promoters_merged_MINUS.fa >
GN20GG_masked_autoXY_promoters_merged_MINUS_REVERSE_Complement
   .fa
```

The fasta files on the plus and minus strand were combined using the cat command, lowercase letters converted to uppercase using the seqret tool from EMBOSS [17], and sequences collapsed into a unique set with `fastx_collapser` [1] .

```
#combine the files
```

```
cat GN20GG_masked_autoXY_promoters_merged_PLUS.fa
GN20GG_masked_autoXY_promoters_merged_MINUS_REVERSE_Complement
   .fa >
GN20GG_masked_autoXY_promoters_merged_TOTAL.fa

#convert to uppercase
seqret GN20GG_masked_autoXY_promoters_merged_TOTAL.fa
GN20GG_masked_autoXY_promoters_merged_TOTAL_UPPER.fa -
   sformat fasta -supper Y

#get unique fasta sequences
fastx_collapser <
   GN20GG_masked_autoXY_promoters_merged_TOTAL_UPPER.fa >
GN20GG_masked_autoXY_promoters_merged_PlusMinus_UNIQUE.fa
```

This yields a file containig 4,113,530 sequences.

1.3 Identifiying a consensus sequence for gRNAs that fall into promoters

The Bioconductor package Biostrings [14] was used to derive a consensus sequence from the list of FASTA sequences generated above as follows (run in R):

```
>library(Biostrings)

>promMINUS<-readDNAStringSet(
"GN20GG_masked_autoXY_promoter_minus_REVERSECOMPLEMENT.fa
   ", format="fasta")

>fm<-consensusMatrix(promMINUS)
```

```
minus<-fm[1:4,]
pwm_minus<-t(t(minus)/rowSums(t(minus)))
```

The following code was used to generate sequence logo plots [6] in R.

```
>library(ggplot2)

>berrylogo<-function(pwm,gc_content=0.5,zero=.0001){
 backFreq<-list(A=(1-gc_content)/2,C=gc_content/2,G=
    gc_content/2,T=
  (1-gc_content)/2)
  pwm[pwm==0]<-zero
bval<-plyr::laply(names(backFreq),function(x){log(pwm[x,])
   -log(
  backFreq[[x]])})
row.names(bval)<-names(backFreq)
p<-ggplot2::ggplot(reshape2::melt(bval,varnames=c("nt","
   pos")),
  ggplot2::aes(x=pos,y=value,label=nt))+
    ggplot2::geom_abline(ggplot2::aes(slope=0), colour = "
       grey",size=2)+
    ggplot2::geom_text(ggplot2::aes(colour=factor(nt)),
       size=8)+
    ggplot2::theme(legend.position="none")+
    ggplot2::scale_x_continuous(name="Position",breaks=1:
       ncol(bval))+
    ggplot2::scale_y_continuous(name="Log relative
       frequency")
  return(p)
}

#invoke the function with:
```

```
berrylogo(pwm_minus, gc_content=0.5, zero=.0001)
```

1.4 Reducing complexity by identifying the most significant clusters

I reasoned that it might be possible to reduce the complexity of a random library by clustering. I defined regions of interest as the 4,671,728 genomic hits of the form GN20GG that fall into or next to known promoter sequences in the human genome with a minimum of 1 bp overlap. I used LCS-HIT (Version 0.5.2) [13] to cluster those regions of interest on the basis of sequence similarity. Clusters were ranked in descending order by the number of members and the top 15 clusters extracted.

```
#cluster sequences based on sequence similarity threshold
# of 0.2 and use the "exact algorithm"
lcs_hit-0.5.21/lcs_hit -i GN20GG_masked_autoXY_promoters
_merged_PlusMinus_UNIQUE.fa -O LCSHIT_OUTPUT20G1 -c 0.2 -g
    1 &
```

LCS-HIT outputs the FASTA-identifiers only, therefore FASTA sequences were extracted using the script `RetrieveFasta.pl`, downloaded from reference [3]. This step is exemplified here for the top cluster (Cluster190).

```
./RetrieveFasta.pl Cluster190
   GN20GG_masked_autoXY_promoters_merged
_PlusMinus_UNIQUE.fa > Cluster190.fa;
```

For each of the top 15 clusters consensus sequences were computed using the Bioconductor package Biostrings (run in R).

```
>library(Biostrings)
>Clust190<-readDNAStringSet("Cluster190.fa", format="fasta
   ")
```

```
>consensusString(Clust190, ambiguityMap=IUPAC_CODE_MAP,
   threshold=0.19,
shift=0L, width=NULL)
```

The threshold option allows the user to define the percentage threshhold at which a given nucleotide will be incorporated into the consensus sequence at a given position. The threshold was varied between 0.14 and 0.25 so as to yield consensus sequences representing roughly 10^4, 10^5 and 10^6 different sequences respectively. The complexity, or number of sequences represented by a cluster, was computed using a C script downloaded from [5] and named `AllSequencesFromConsensus.c`.

```
#compile with
gcc -o AllSequencesFromConsensus AllSequencesFromConsensus
   .c

#run the script and pipe the output to line-count (as
# shown for the consensus sequence of Cluster190_T20)
./AllSequencesFromConsensus RRRGRGRRRRRGRRGRRGV | wc -l
```

Next, the "GenomeSearch" function of the Biostrings package (see Section 1.1) was used to run each of the consensus sequences for each of the top 15 clusters against the masked human genome.

```
#store the sequences that should be run against the genome
# in a DNAStringSet object dict0, here shown for
   Cluster190
>dict0<-DNAStringSet(c("GRRRGRGRRRRRGRRGRRGVNGG", "
   GRRRRRRRRRRRRRRGRRGVNGG",
"GVRRRRRRRRRRRRRRRRSVNGG", "GVRRRRRRRRRRRRRRRRSVNGG",
"GVVRRRRRRRRRRRRRRVVVNGG", "GVVVRRRRRRRRRRRRVVVVNGG"))

#provide an identifier for each of the consensus sequences
   ,
#here, cluster number and chosen threshold were used,
#again shown for Cluster190
>names(dict0)<-c("Cluster190_T20", "Cluster190_T19", "
   Cluster190_T18", "Cluster190_T17", "Cluster190_T16", "
   Cluster190_T15")

#invoke as follows:
>GenomeSearch_masked(dict0, outfile="
   Top15Clusters_masked_allregions.txt")
```

Because gRNAs are known to tolerate mismatches in their target sites [11], I reasoned that counting only exact matches probably underestimates the true number of target sites of any given sequence. Thus, the analysis was repeated allowing for an arbitrary number of up to 3 mismatches (gRNAs are known to tolerate more mismatches, especially towards the 5' end of the protospacer sequence. However, position of the mismatch in the gRNA was not taken into account here).

```
# allowing for a maximum of three mismatches
# change relevant parameter in the "GenomeSearch" function
#of the code

>plus_matches <- matchPattern(pattern, subject, max.
   mismatch=3, min.mismatch=0, fixed=c(pattern=FALSE,
   subject=TRUE))

>runAnalysis_masked_3mismatch(dict0, outfile="
   Top15Clusters_masked_allregions_3mismatch.txt")
```

Output files from both scripts were processed as follows and the results stored in Table 3.1 on p. 96 of the PhD thesis.

```
# retrieve hits for autosomes and sex chromosomes only
grep "Cluster190_T20" Top15Clusters_masked_allregions.txt
```

```
    | grep -v 'random' | grep -v 'hap' | grep -v 'chrUn' |
    grep -v chrM | awk '{print $1 "\t" $2 "\t" $3}' >
   Cluster190_T20_autoXY

#count number of hits that fall into promoter regions
intersectBed -a Cluster190_T20_autoXY -b annotation_files/
   promoters_merged.bed -wa -wb | sortBed | wc -l;

#count the number of unique promoter regions hit
intersectBed -a  Cluster190_T20_autoXY -b
   annotation_files/promoters_merged.bed -wb | cut -f 5-8
   | sortBed | uniq | wc -l;
```

I defined the targeting efficiency for each of the possible consensus sequences of a given cluster by dividing the number of promoter hits by the total number of unique sequences (complexity) of the consensus. For each cluster the consensus sequences with the highest targeting sequence was chosen. The 15 top clusters were then ranked by targeting efficiency and the top 6 clusters chosen for library preparation.

Chapter 2

Analysis of the data generated by sequencing the degenerate and MNase digest libraries

2.1 Read trimming and quality filtering

Addition of sequencing adapters was non-directional, which means the reads contain gRNA cassettes both in forward and reverse direction.

First, the gRNAs that were sequenced in the forward direction were extracted using cutadapt [12] to trim away the plasmid sequence (shown here for the fastq sequencing data file generated from the Random N19 library).

```
cutadapt -g
   AAGTATTTCGATTTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACC -O
   41 -m 2 --untrimmed-output=Untrimmed255_R1_50F.fastq.gz
    -o 255-N19_R1_F50_trimmed.fastq.gz ../255-
   N19_S5_L001_R1_001.fastq.gz;

cutadapt -a GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCTAGTCCG -O
   33 -m 2 --untrimmed-output=Untrimmed255_R1_19R.fastq.gz
    -o 255-N19_R1_F50+19R_trimmed.fastq.gz 255-
   N19_R1_F50_trimmed.fastq.gz;
```

Next, the reads stored in the untrimmed output, which will contain reads of the cassette sequenced in the reverse direction were concatenated, sorted and gRNA sequences extracted as follows (again showing example of reads from the N19 Random library):

```
cat Untrimmed255_R1_50F.fastq.gz Untrimmed255_R1_19R.fastq
   .gz > Untrimmed255_R1_50F+19R.fastq.gz;

zcat Untrimmed255_R1_50F+19R.fastq.gz | paste - - - - |
   sort -k1,1 -t " " | tr "\t" "\n" > Untrimmed255_R1_50F
   +19R_sorted.fastq;

cutadapt -a
```

```
   GGTGTTTCGTCCTTTCCACAAGATATATAAAGCCAAGAAATCGAAATACTT -O
   41 -m 2 --untrimmed-output=Untrimmed255_50FRC.fastq.gz
   -o 255_R1_50FRC_trimmed.fastq.gz Untrimmed255_R1_50F+19
   R_sorted.fastq;

cutadapt -g CGGACTAGCCTTATTTTAACTTGCTATTTCTAGCTCTAAAAC -O
   33 -m 2 --untrimmed-output=Untrimmed255_19RRC.fastq.gz
   -o 255_R1_50FRC+19RRC_trimmed.fastq.gz 255
   _R1_50FRC_trimmed.fastq.gz;
```

This results in extraction of gRNA sequences sequenced in the reverse direction.

```
### Double untrimmed reads:
cat Untrimmed255_50FRC.fastq.gz Untrimmed255_19RRC.fastq.
   gz > Untrimmed255_50FRC+19RRC.fastq.gz;

zcat Untrimmed255_50FRC+19RRC.fastq.gz | paste - - - - |
   sort -k1,1 -t " " | tr "\t" "\n" > Untrimmed255_50FRC
   +19RRC_sorted.fastq;

### Successfully trimmed reads:

cat 255-N19_R1_F50+19R_trimmed.fastq.gz 255_R1_50FRC+19
   RRC_trimmed.fastq.gz > 255_R1_4waytrimmed.fastq.gz;

zcat 255_R1_4waytrimmed.fastq.gz | paste - - - - | sort -
   k1,1 -t " " | tr "\t" "\n" > 255_R1_4waytrimmed_sorted.
   fastq;

### Extracted gRNA sequences sequenced in reverese
   direction need to be reverse-complemented
```

```
zcat 255_R1_50FRC+19RRC_trimmed.fastq.gz |
   fastx_reverse_complement -Q33 -z > 255_R1_50FRC+19
   RRC_trimmed_onedirection.fastq.gz;


cat 255-N19_R1_F50+19R_trimmed.fastq.gz 255_R1_50FRC+19
   RRC_trimmed_onedirection.fastq.gz > 255
   _R1_4waytrimmed_onedirection.fastq.gz;

zcat 255_R1_4waytrimmed_onedirection.fastq.gz | paste - -
   - - | sort -k1,1 -t " " | tr "\t" "\n" > 255
   _R1_4waytrimmed_sorted_onedirection.fastq;
```

Next, read length distribution between 15 and 40 bp were plotted for each of the libraries (again shown here for the file containing reads from the N19 Random library):

```
awk '{y= i++ % 4 ; L[y]=$0; if(y==3 && length(L[1])<=40) {
   printf("%s\n%s\n%s\n%s\n",L[0],L[1],L[2],L[3]);}}' 255
   _R1_4waytrimmed_sorted.fastq > 255
   _R1_4waytrimmed_sorted_smaller40.fastq;

awk '{if(NR%4==2) print}' 255
   _R1_4waytrimmed_sorted_smaller40.fastq | awk '{print
   length($1)}' | sort -n >  255
   _R1_4waytrimmed_sorted_smaller40_lengths;
```

Per base sequence quality after trimming was assessed using FASTQC [2]

```
fastqc 255_R1_4waytrimmed_sorted_smaller40.fastq  --outdir
   =../FASTQC_Trimmedreads
```

2.2 Histogram of gRNA lengths

This code was used to generate the plots in Figure 3.12 on page 100 of the PhD thesis. Histograms of read lengths were generated in R:

```
reads<-read.table("255
   _R1_4waytrimmed_sorted_smaller40_lengths", header=F)

par(mar=c(5,6+2,4,2)+1)
hist(reads[,1], breaks=seq(0,40,by=1), col="grey", main="
   Method 1 - Random N19", xlab="Insert length (bp)", ylim
   =c(0, 500000), las=1, ylab="", cex.main=1.3, cex.axis
   =1.3, cex.lab=1.3)
title(ylab = "Frequency", line = 6, cex.lab=1.3)
```

2.3 Histogram of read frequencies

This code was used to generate the plots in Figure 3.11 on page 99 of the PhD thesis.

```
awk '{if(NR%4==2) print}' 255
   _R1_4waytrimmed_sorted_onedirection_smaller40.fastq |
   sort | uniq -c | sort | sed 's/^ *//' > 255
   _R1_4waytrimmed_sorted_onedirection_smaller40_sortedbyfrequency
   ;

#Histograms of read frequency were generated in R:
reads<-read.table("255
   _R1_4waytrimmed_sorted_onedirection_smaller40_sortedbyfrequency
   ", sep=" ", head=F)
colnames(reads)<-c("counts", "sequence")
reads_upto5<-reads[reads$counts<=5,]
```

```
reads_greater5<-reads[reads$counts>=5,]
par(oma=c(0,0,0,0))
par(mar=c(6,6,8,3))
par(fig=c(0.1,1,0,0.75))
plot(rownames(reads), reads$counts, type="p", xlab="gRNA
   ranked by counts", xlim= c(0, 600000), ylim=c(5,350),
   ylab="", las=1, col="orange", lwd=4, cex.axis=1.7, cex.
   lab=1.9)
title(ylab = "Read counts", line = 4, cex.lab=1.9)
par(fig=c(0.1,1,0.45,1), new=TRUE)
plot(rownames(reads_upto5), reads_upto5$counts, type="l",
   main="Method 3 - Mouse", xlab="", xlim= c(0, 600000),
   ylim=c(0,5), ylab="", lwd=8, col="orange", las=1, cex.
   axis=1.7, cex.lab=1.9, cex.main=1.9)
title(ylab = "Read counts", line = 4, cex.lab=1.9)
```

2.4 Alignment to the reference genome

Trimmed reads were aligned to the reference genome, either human (GRCh37/hg19) or mouse (NCBI37/mm9) as appropriate, using BWA [9], again shown using the file containing reads from the N19 Random library as an example.

```
#use bwa to align reads to human genome without allowing
   mismatches and printing all alignments, seed length is
   set to 19

bwa aln -n 0 -o 0 -l 19 -N /mnt/store2/local_data/
   genomic_data/human/GRCh37/human_GRCh37.tmp 255
   _R1_4waytrimmed_sorted.fastq > 255_R1_4waytrimmed.sai
```

```
#generate the sam file
bwa samse -n 10000 /path2/human_GRCh37.tmp 255
   _R1_4waytrimmed.sai 255_R1_4waytrimmed_sorted.fastq >
   255_R1_4waytrimmed.sam
```

Uniquely mapped reads were counted using Samtools [10].

```
samtools view -S -q1 255_R1_4waytrimmed.sam | wc -l
```

The remainder of the analysis (parts of which are shown in Fig. 3.18 on page 111 of the thesis) was conducted by my collaborator Karolina Worf at the Helmholtz Institute in Munich. The documentation for this analysis is available at hgmubox (`https://hmgubox.helmholtz-muenchen.de:8001/d/6c6e75236e/`; password: Coralina).

Chapter 3

Design of the EMT5000 library

3.1 Defining regions of interest

A list of genes known to be involved in the regulation of epithelial-to-mesenchymal transition (EMT) was taken from DeCraene et al. [7]. Regions of interest within the promoter of these genes were identified manually by inspection of chromatin marks in UCSC genome browser (**table 3.1**).

Chromosome	Start	Stop	Gene
chr1	170626538	170637878	PRRX1
chr2	145272896	145282545	ZEB2
chr2	145310788	145311630	ZEB2
chr6	166578775	166584033	Brachyury (T gene)
chr6	166586466	166588249	Brachyury (T gene)
chr7	19155427	19162115	TWIST1
chr8	49831094	49838789	SLUG
chr10	31549929	31552360	ZEB1
chr10	31603262	31611019	ZEB1
chr14	61113258	61126351	SIX1
chr14	95235188	95236645	GSC (Goosecoid)
chr16	68765472	68768900	Cdh1
chr16	68770501	68779468	Cdh1
chr16	86596822	86601033	FOXC2
chr18	25616470	25616815	Cdh2
chr18	25753143	25759002	Cdh2
chr18	25763319	25764152	Cdh2
chr18	25783828	25784775	Cdh2
chr18	52966479	52970132	TCF4
chr18	52983584	52991765	TCF4
chr18	52994740	52997201	TCF4
chr18	53067559	53071402	TCF4
chr18	53072594	53073776	TCF4
chr18	53087724	53090359	TCF4
chr18	53176467	53178784	TCF4
chr18	53252816	53257791	TCF4
chr18	53259747	53260381	TCF4
chr18	53301547	53303603	TCF4
chr19	1631468	1633670	E47/TCF3
chr19	1646514	1653855	E47/TCF3
chr19	1655335	1656204	E47/TCF3
chr19	1660918	1661677	E47/TCF3
chr20	48592707	48600991	SNAIL1
chrX	56258091	56260688	KLF8

Table 3.1: Regions of interest around the promoter regions of 15 genes known to be involved in the regulation of EMT [7]

The genomic regions listed in **table 3.1** where chosen as target sites for the design of gRNAs and saved in the file `EMT_genepromoter_comprehensive.gff`. All gRNAs falling into these regions were then found using the Bedtools [16] intersect function on the file containing all gRNAs with an NGG PAM found in the genome (see section

1.1 for how this file was generated).

```
intersectBed -a GN20GG_masked_autoXY.gff -b
   EMT_genepromoter_comprehensive.gff
-f 1 -wa -wb >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive.gff
```

Guide RNAs were then aligned to the genome in order to identify those that map uniquely. Before alignment, files had to be converted into the correct format as follows:

```
##for alignment need to first convert file into fasta
   format
##separate + and - strand
cut -f 1,2,3,4
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive.gff
    | grep '+' | awk '{print "chr" $1 ":" ($2-1) "-" $3}'
   >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_PLUS
   .2bit;

cut -f 1,2,3,4
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive.gff
    | grep -v '+' | awk '{print "chr" $1 ":" $2 "-" $3}' >

   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_MINUS
   .2bit;


##convert to FASTA format using twoBitToFa
twoBitToFa /path2/human/GRCh37/hg19.2bit
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_PLUS
   .fa seqList=
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_PLUS
   .2bit;

twoBitToFa  /path2/human/GRCh37/hg19.2bit
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_MINUS
   .fa seqList=
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_MINUS
   .2bit;

##create reverse complement of guide RNAs on the minus
##strand using the script reverse_complement_fasta.py

python reverse_complement_fasta.py
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_MINUS
   .fa >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_MINUS_RC
   .fa;

##combine the files for the + and - strand
cat
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_PLUS
   .fa
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_MINUS_RC
   .fa >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive.fa;
```

The PAM sequence was removed using `trimming.py` and alignment to the genome without the PAM (i.e. as GN19 instead of GN20GG).

```
bwa aln -n 0 -o 0 -l 10 -N -I ~/path_to/GRCh37/
   human_GRCh37.tmp
GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_noPAM.
   fa >
```

```
GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_noPAM.
   sai;

bwa samse -n 10000 ~/path_to/GRCh37/human_GRCh37.tmp
GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_noPAM.
   sai
GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_noPAM.
   fa >
GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_noPAM.
   sam;

samtools view -S -q1
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_noPAM
   .sam | cut -f  1 | awk -F ':' '{print $1 "\t" $2}' |
   awk -F '-' '{print $1 "\t" $2 "\t" $3}' | sortBed |
   uniq >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_noPAM_u
   .bed
```

This file contains 5086 sequences, i.e. 5086 gRNA sequences of the form GN19 that align uniquely to the genome and are followed by an NGG PAM and are found in the regions of interest.

For cloning into the gRNA vector pgRNA-pLKO.1 (see Methods) by Gibson cloning, vector-derived sequences were attached to either end of the gRNA sequence. To this end the FASTA files for the final set of 5086 unique gRNAs were retrieved and sequences appended as follow:

```
#split according to + and - strand

grep '+'
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
```

```
    noPAM_unique_strand.bed | awk '{print $1 ":" $2 "-" $3
   }' >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_PLUS.2bit

grep -v '+'
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand.bed | awk '{print $1 ":" $2 "-" $3
   }' >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_MINUS.2bit

#convert to FASTA format and reverse-complement sequences
   on the
#minus strand

twoBitToFa /path2/human/GRCh37/hg19.2bit
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_PLUS.fa seqList=
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_PLUS.2bit;

twoBitToFa /path2/human/GRCh37/hg19.2bit
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_MINUS.fa seqList=
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_MINUS.2bit

python reverse_complement_fasta.py
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_MINUS.fa >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_
    noPAM_unique_strand_MINUS_RC.fa

#merge the two files

cat
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_

noPAM_unique_strand_PLUS.fa
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_

noPAM_unique_strand_MINUS_RC.fa >
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_

noPAM_unique_strand.fa

# use trimming.py to remove PAM (because coordinates
   extracted from SAM file contained PAM, while alignment
   was run without PAM) and paste the vector sequences for
    Gibson cloning on either side of the gRNA sequence

sed 'n; s/$/GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCT/'
   GN20GG_masked_autoXY_EMT_genepromoter_
   comprehensive_complete_noPAM_unique.fa | sed 'n; s/^/
   TCTTGTGGAAAGGACGAAACACC/g' | paste - - | awk '{print $2
    "\t" $1}' > EMT_guides_Custom_Array.txt
```

A pool of sequences of the form 5'-TCTTGTGGAAAGGACGAAACACC-GN19-GTTTTAGAGCT AGAAATAGCAAGTTAAAATAAGGCT-3', where GN19 donates the 5086 different guide sequences was then ordered from Custom Array Inc.

Chapter 4

Analysis of screen with the EMT5000 library and stable cell lines expressing the dCas9-SET7 or dCas9-p300 chromatin modifier respectively

4.1 Read trimming and Quality Filtering

gRNA sequences integrated into the genome of FACS-sorted cells were amplified using PCR to add Illumina Nextera adapters. Libraries were sequenced on the Illumina HiSeq instrument. The resulting sequencing reads have the following general structure:

`NNNNNNNAAGTATTTCGATTTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACCG-N19-GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCTAGTCCGNNNNNNN`

The first 7 N bases are the 5' barcode, followed by the plasmid 'stuffer'. GN19 denotes the gRNA sequence from the EMT5000 library, which is followed by another plasmid sequence and the last 7 N bases are the 3' barcode. The 5' and 3' barcodes serve as unique molecular identifiers (UMIs), allowing counting of original gRNA sequences extracted from lentivirus-infected cells by removing PCR-amplification bias.

Sequences from Lane1 and Lane2 of the flow cell were combined using the unix 'cat' command. Next, the 5' barcode was extracted from the reads using cutadapt (version 1.2.1) [12], requiring a minimum overlap of 35 bp between the plasmid stuffer sequence and the read with a maximum error of 10 % and a minimum barcode length of 7 bp.

```
Command line parameters:
-a AAGTATTTCGATTTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACC -O
   35 -m 7

# example bash loop to run cutadapt over all fastq files
# in the directory
for i in *.fastq.gz
do cutadapt -a
   AAGTATTTCGATTTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACC -O
   35 -m 7 --untrimmed-output="${i%.fastq.gz}"
   _5bc_untrimmed.fastq.gz -o "${i%.fastq.gz}"_5bc.fastq.
   gz "$i";
done
```

The 3' barcode was retrieved from the read in an analogous way:

```
Command line parameters:
-g GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCTAGTCCG -O 28 -m 7

# example bash loop to run cutadapt over all fastq files
# in the directory
for i in *.fastq.gz
do cutadapt -g GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCTAGTCCG
   -O 28 -m 7 --untrimmed-output="${i%.fastq.gz}"
   _3bc_untrimmed.fastq.gz -o "${i%.fastq.gz}"_3bc.fastq.
   gz "$i";
done
```

Finally, the gRNA sequence was extracted from the read, requiring a minimum length of 2 bp (to discard reads that contain no gRNAs and are derived from primer dimers):

```
Command line parameters:
cutadapt -g
   AAGTATTTCGATTTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACC -a
   GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCTAGTCCG -n 2 -m 2 -O
    10

# example bash loop to run cutadapt over all fastq files
# in the directory
for i in *.fastq.gz
do cutadapt -g
   AAGTATTTCGATTTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACC -a
   GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCTAGTCCG -n 2 -m 2 -O
    10 --untrimmed-output="${i%.fastq.gz}"_gRNA_untrimmed.
   fastq.gz -o "${i%.fastq.gz}"_gRNA.fastq.gz "$i";
done
```

Next, the barcode and gRNA reads were quality-filtered using `fastq_quality_filter` from the fastx-toolbox [1]. Reads where any base has a Quality score of less than 20 were discarded (Example code below is for the gRNA reads.)

```
Command line parameters: -Q33 -q 20 -p 1

#loop over all files in directory:
for i in *_gRNA.fastq; do fastq_quality_filter -Q33 -q 20
   -p 1 -i "$i" -o "${i%_gRNA.fastq}"_gRNA_Q20.fastq; done
```

Subsequently files were converted from fastq to fasta format using `fastq_to_fasta` from the fastx-toolbox [1].

```
for i in *_gRNA_Q20.fastq;
do fastq_to_fasta -Q33 -n -i  "$i" -o "${i%gRNA_Q20.fastq
   }"gRNA_Q20.fasta;
done
```

4.2 Alignment to the EMT5000 library

Guide RNA reads were next aligned back onto the indexed EMT5000 reference library using bwa (version: 0.6.2-r126) [9].

The indexed EMT5000 library file used as a reference for alignment and was derived from the file `GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_noPAM_unique_s` (see section 3.1) using `bwa index`:

```
bwa index
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_complete_noPAM_u
   .fa
```

I empirically tested the following alignment parameters:

```
#Command line options no mismatches, no indels:
bwa aln -n 0 -o 0 -l 5 -N -I

#Command line options 2 mismatches, no indels:
bwa aln -n 2 -o 0 -l 5 -N -I

#Command line options 3 mismatches, no indels:
bwa aln -n 3 -o 0 -l 5 -N -I

#Command line options 2 mismatches and default open gaps
   (1):
bwa aln -n 2 -l 5 -N -I

#Command line options 3 mismatches and default open gaps
   (1):
do bwa aln -n 3 -l 5 -N -I
```

I found that allowing 2 mismatches and 1 gap gave the best alignment (see also the table of read numbers included in the Appendix of the PhD thesis). The alignment was invoked as follows:

```
for i in ../*_Q20.fasta;
do bwa aln -n 2 -l 5 -N -I EMT5000_library_reference/
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_
complete_noPAM_unique_strand_PAMremoved.fa "$i" > "${i%Q20
   .fasta}"Q20_aligned_2mismatches1gap.sai;
done

for i in *Q20_aligned_2mismatches1gap.sai;
do bwa samse -n 10000 EMT5000_library_reference/
   GN20GG_masked_autoXY_EMT_genepromoter_comprehensive_
```

```
complete_noPAM_unique_strand_PAMremoved.fa "$i" "${i%
   Q20_aligned_2mismatches1gap.sai}"Q20.fasta > "${i%
   Q20_aligned_2mismatches1gap.sai}"
   Q20_aligned_2mismatches1gap.sam;
done
```

Following alignment allowing 2 mismaches and 1 gap, reads that mapped uniquely to the forward strand were extracted using Samtools [10]:

```
samtools view -F 20 -q 1 -S aligned_gRNA.sam  >
   gRNA_uniquely_mapped.sam
```

To check how many different gRNAs from the library were sequenced in each sample, use:

```
samtools view -F 20 -S  aligned_gRNA.sam | cut -f 3 | sort
    | uniq | wc -l
```

4.3 Combining the gRNA and adapter sequences in a single file

For subsequent analysis it was necessary to construct a tab-separated file with 3 columns containing the FASTA identifier (read-ID), gRNA chr:start-end and barcode sequence for each read.

```
# extract the gRNA:
samtools view -F 20 -q 1 -S aligned_gRNA.sam | awk '{
   print$1 ".\t" $3}'  > gRNA_uniquelymapped

# extract the FASTA identifier:
samtools view -F 20 -q 1 -S aligned_gRNA.sam | awk '{
   print$1 "."}' > gRNA_uniquelymapped_readID
```

The files containing the quality-filtered 5' and 3' barcode (UMI sequence) generated above, were modified as follows (shown for 5' barcode only):

```
for i in *_5bc_Q20.fasta
do cat "$i" | paste - - | awk -F ' ' '{print $1 ".\t" $3}'
    > "${i%_5bc_Q20.fasta}"_5bc_Q20_point.fasta;
done
```

Next, only the barcodes associated with gRNAs that aligned uniquely were retrieved using grep:

```
for i in *_5bc_Q20_point.fasta
do grep -wFf "${i%_5bc_Q20_point.fasta}"
   gRNA_uniquelymapped_readID "$i" > "${i%_5bc_Q20_point.
   fasta}"gRNA_uniquelymapped_readID_with_5bc;
done
```

For each read, identified by its readID, the gRNA sequence and 5' and 3' barcodes were combined into a single file. The three files were joined (finding the union) using the JOIN command, which requires the files to be sorted in the following way:

```
for i in *gRNA_uniquelymapped;
do awk -F "\t" '{print ">" $1 "\t" $2}' "$i"| sort -k 1b,1
    > "${i%}"_sorted;
done

for i in *gRNA_uniquelymapped_readID_with_3bc;
do sort -k 1b,1 "$i" > "${i%}"_sorted;
done

for i in *gRNA_uniquelymapped_readID_with_5bc; do sort -k
   1b,1 "$i" > "${i%}"_sorted;
done
```

Next, the three files were joined as follows:

```
for i in *gRNA_uniquelymapped_sorted;
do join "$i" "${i%gRNA_uniquelymapped_sorted}"
   gRNA_uniquelymapped_readID_with_5bc_sorted | join - "${
   i%gRNA_uniquelymapped_sorted}"
   gRNA_uniquelymapped_readID_with_3bc_sorted > "${i%
   gRNA_uniquelymapped_sorted}"
   gRNA_uniquelymapped_readID_gRNA_5bc_3bc_length14;
done
```

This yields a file containing for each uniquely mapped read its readID, the gRNA it mapped to (chr:start-stop) and the UMI found in the read (of length exactly 14 bp).

4.4 Deriving gRNA counts from UMI-barcodes without PCR error correction

The gRNA counts were derived by counting the number of times each gRNA occurs together with each barcode, which acts as a unique molecular identifier (UMI). To do the gRNA counting without any error correction, the script `collapse_barcodes.py` was run using a maximum edit distance of 0, i.e. not correcting any PCR errors that might have occurred in the barcode. This script can be invoked as follows:

```
python collapse_barcodes.py Inputfilename 0
```

The script accepts a tab-separated file with columns for (1) read-ID, (2) gRNA (chr:start-stop), (3) barcode and outputs two outputfiles with extension `_frequency_raw` and `frequency_no_orphans`. In the latter output orphan barcodes, i.e. barcodes that are only present in a single read, were removed prior to gRNA counting. Each output file has two comma-separated columns listing (1) the gRNA (chr:start-stop) and (2) the gRNA count.

The script was run over all files with the extension `_uniquelymapped_readID_gRNA_5bc_3bc_length14` as follows:

```
for i in *length_14;
do python collapse_barcodes.py "$i" 0;
done
```

4.4.1 Assessing PCR error by plotting the number of reads per gRNA against number of different gRNA sequences

To assess whether PCR error drives barcode diversity, I treated the gRNA part of the read like a barcode and plotted the correlation between number of reads and counts (see Figure 4.8. on page 134 of the PhD thesis and also section 4.7 below). This assumes that the likelihood of introducing an error into the sequence is the same for the UMI barcodes and gRNA portions of the amplicon.

The barcode sequence was replaced with the gRNA sequence for each read to generate a tab-separated file with columns [0] readID [1] gRNA chr:start-stop [2] 'pseudo-barcode'(gRNA sequence) as follows:

```
#get gRNA sequence
cat Sample_gRNA_Q20.fasta | paste - - | awk -F ' ' '{print
    $1 ".\t" $3}' | sort > Sample_gRNA_Q20_point

#get the read ID
awk -F '\t' '{print $1}' Sample_gRNA_5bc_3bc_length14 |
   sort > Sample_gRNA_5bc_3bc_length14_read_ID

#get gRNA sequence for each readID
grep -wFf Sample_gRNA_5bc_3bc_length14_read_ID
   Sample_gRNA_Q20_point >
   Sample_gRNA_Q20_point_uniquely_mapped
```

```
#sort the previously generate file containing [0] readID,
   [1] gRNA chr:start-stop, [2] UMI of 5' and 3' barcode
sort -k 1b,1
   Sample_gRNA_uniquelymapped_readID_gRNA_5bc_3bc_length14
    > Sample_gRNA_uniquelymapped_
readID_gRNA_5bc_3bc_length14_sorted

#sort the gRNA sequence file
sort -k 1b,1 Sample_gRNA_Q20_point_uniquely_mapped >
   Sample_gRNA_Q20_point_uniquely_mapped_sorted

#join the two files on readID
join
   Sample_gRNA_Q20_aligned_2mismatches1gap_uniquelymapped_
readID_gRNA_5bc_3bc_length14_sorted
   Sample_gRNA_Q20_point_uniquely_mapped_sorted | awk -F '
    ' '{print $1 "\t" $2 "\t" $4}' >
   Sample_gRNA_Q20_aligned_2mismatches1gap_uniquelymapped_
readID_gRNA_instead_of_barcode
```

This file was then run through `collapse_barcodes.py` as above and results were plotted as described in section 4.7.

```
python collapse_barcodes.py
   Sample_gRNA_Q20_aligned_2mismatches1gap_
uniquelymapped_readID_gRNA_instead_of_barcode 0
```

4.5 Deriving gRNA counts from UMI-barcodes with naive PCR error correction

The script `collapse_barcodes.py` was run using a maximum edit distance of 4.

```
python collapse_barcodes.py Samplefilename 4
```

This means that before counting how many different barcodes are associated with each gRNA, the barcodes are collapsed into groups. Barcodes are first ranked in decreasing order based on the number of reads harbouring its sequence. The barcode with the most reads forms the first group. If the second-ranked barcode is within 4 edit-distances of this barcode it will be assumed to have originated by PCR error and will be added to the group. If the barcode differs from the group by greater than 4 edits, it will form its own group and so on. The number of groups per gRNA is the count after error correction. This reflects the number of original gRNA-barcode combinations.

The script was run over all files with the extension `_uniquelymapped_readID_gRNA_5bc_3bc_length14` as follows:

```
for i in *length_14;
do python collapse_barcodes.py "$i" 0;
done
```

4.6 Bayesian PCR error correction of barcoded sequencing data

A Bayesian error correction script was written by James E. Barrett to infer gRNA counts from the UMI data. The model takes into account the fact that the 14 bp UMI consists of a 5' and 3' barcode that was attached to the gRNA amplicon during and initial round off PCR amplification during the sequencing library prep. The model infers the most likely number of initial gRNA-barcode data given the barcode sequences observed in the sequencing sample.

This Bayesian model takes as input the number of reads associated with each gRNA-UMI combination (without PCR error correction). I calculated these using the script make-csv-4Bayes.py. The script was run over all samples as follows:

```
for i in *length14; do python ../make_csv_4Bayes.py "$i";
   done
```

This script again takes the tab-separated three column file that lists (1) read ID, (2) gRNA (chr:start-end) and (3) barcode consisting of 5' UMI and 3' UMI fused together as input. The output is a csv file with three columns containing (1) gRNA (chr:start-end), (2) barcode consisting of 5' UMI and 3' UMI fused together and (3) number of reads associated with each barcode. The outputfile has the extension `_barcode_readcounts.csv` and is fed into a Bayesian error correction script described below.

4.6.1 Bayesian PCR error correction of barcoded sequencing count data script by James E. Barrett

This script and documentation was kindly contributed by James E. Barrett.

The Bayesian model infers the number of unique original barcoded gRNA molecules from noise-corrupted count data. The model estimates a corrected read count, which may be interpreted as a proxy for the original noise-free number of unique barcodes associated with a particular gRNA.

Model definition

For each gRNA we observe N barcode pairs denoted by $(\mathbf{y}_i^1, \mathbf{y}_i^2)$ where the superscript denotes the first and second barcodes and $i = 1, \ldots, N$. Elements of the

d-dimensional vector $\mathbf{y}_i^\eta \in \{\mathtt{T}, \mathtt{C}, \mathtt{G}, \mathtt{A}\}^d$ where $\eta = [1, 2]$. The number of corresponding sequencing reads is denoted by $\sigma_i \in \mathbb{Z}_+$.

The model assumes that there exist Q *latent barcodes* $\mathbf{x}_1^\eta, \ldots, \mathbf{x}_Q^\eta$ from which the observed barcodes are generated in a noise corrupting stochastic process (PCR amplification errors and random barcode switching). The model further assumes that for each pair $(\mathbf{y}_i^1, \mathbf{y}_i^2)$ only one of the observed barcodes is written in terms of the latent barcode via

$$\mathbf{y}_i^\eta = \sum_{q=1}^{Q} w_{iq}^\eta \theta(\mathbf{x}_q^\eta) \quad \text{subject to} \quad w_{iq}^\eta \in [0,1] \quad \text{and} \quad \sum_{q,\eta} w_{iq}^\eta = 1. \tag{4.1}$$

There is therefore only one non-zero value of $[\mathbf{w}_i^1, \mathbf{w}_i^2]$ that indicates which latent barcode the observed pair is associated with. The function θ represents a noise corrupting stochastic process where the status of each nucleotide site may be changed randomly with probability $\beta \in [0, 1/2]$. We can therefore write

$$p(y_{i\mu}^\eta | x_{q\mu}^\eta, \beta) = \begin{cases} (1-\beta)\delta_{y_{i\mu}^\eta x_{q\mu}^\eta} + \beta(1 - \delta_{y_{i\mu}^\eta x_{q\mu}^\eta}) & \text{if } w_{iq}^\eta = 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

for $\mu = 1, \ldots, d$. We denote the collections of $\mathbf{x}_q^\eta$, $\mathbf{y}_i^\eta$ and $\mathbf{w}_i^\eta$ by $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{W}$ respectively. The posterior is

$$p(\mathbf{X}, \mathbf{W} | \mathbf{Y}, \boldsymbol{\sigma}, \beta) \propto p(\mathbf{Y} | \mathbf{X}, \mathbf{W}, \boldsymbol{\sigma}, \beta) p(\mathbf{X}) p(\mathbf{W}) \tag{4.3}$$

with

$$p(\mathbf{Y} | \mathbf{X}, \mathbf{W}, \boldsymbol{\sigma}, \beta) = \prod_i \left[\sum_{q,\eta} w_{iq}^\eta p(\mathbf{y}_i^\eta | \mathbf{x}_q^\eta, \beta) \right]^{\sigma_i}. \tag{4.4}$$

Maximum entropy priors for $\mathbf{X}$ and $\mathbf{W}$ are uniform distributions so $p(\mathbf{X})$ and $p(\mathbf{W})$ are constant.

Inference of model parameters

The Maximum A Posteriori (MAP) solution of $\mathbf{W}$ is denoted by $\mathbf{W}^*$. Since only one element of $[\mathbf{w}_i^1, \mathbf{w}_i^2]$ is non-zero the expression (4.4) is maximised by selecting $\text{argmax}_{q,\eta} p(\mathbf{y}_i^\eta | \mathbf{x}_q^\eta, \beta)$ as the non-zero element.

To find the MAP solution for nucleotide μ of the latent barcode indexed by (q, η) we consider all observed barcodes that generated from it (as defined by $\mathbf{W}$). If we let n_1 and n_0 denote the total number of matches and mismatches respectively between that latent barcode and the associated observed barcodes, then the corresponding data likelihood is $(1-\beta)^{n^1_{q\mu}} \beta^{n^0_{q\mu}}$. This will be maximised if the number of matches is maximised. This is achieved selecting the most common observed nucleotide as the value for the latent nucleotide (while taking into account multiple counts).

If we let N_1 and N_0 denote the total number of matches and mismatches respectively across all of the latent barcodes and observed data then we can write

$$\log p(\mathbf{Y}|\mathbf{X}, \mathbf{W}, \beta) = N_1 \log(1-\beta) + N_0 \log \beta. \tag{4.5}$$

It is straightforward to show that the MAP estimate for beta is

$$\beta = \frac{N_0}{N_0 + N_1}. \tag{4.6}$$

The optimisation subroutine is initialised as follows:

Cluster into Q groups based on the *Hamming distance* between two barcodes (the Hamming distance is equivalent to the *edit distance*):

$$h(\mathbf{y}_i, \mathbf{y}_j) = \frac{1}{d} \sum_{\mu=1}^{d} \delta_{(1-y_{i\mu}) y_{j\mu}}. \tag{4.7}$$

The corrected read counts are inferred as follows:

For a given value of Q we denote the value of the likelihood (4.4) at the MAP parameter estimate by

$$L(Q) = p(\mathbf{Y}|\mathbf{X}^*, \mathbf{W}^*, \beta^*). \tag{4.8}$$

The *Bayes information criterion* (BIC) score is defined by

$$\mathrm{BIC}(Q) = -2 \log L(Q) + 2dQ \log N \tag{4.9}$$

where $2dQ$ is the number of free parameters in the model. The *corrected read count* is defined by

$$Q^* = \mathrm{argmin}_Q \mathrm{BIC}(Q). \tag{4.10}$$

Bayesian error correction script: The Code

The analysis is performed in R:

```
library(reshape2)

### Load and prepare a data file

# Length of barcode
D <- 7
# Load up one of the data files (needs to be in the
   current directory)
data <- read.csv("Samplename_length14_barcode_readcounts.
   csv",header=FALSE)
# vector of all the unique gRNA names
gRNA <- unique(data$V1)
# Total number of unique gRNAs
G <- length(gRNA)

### Generate datasets of barcodes
```

```
# Preallocate a list structure to hold the barcode
   datasets
Y <- vector('list',G)

# This loop goes through each gRNA, pulls out all the
   associated barcodes and puts them in a character matrix
for(mu in 1:G){
   ind <- which(data[[1]]==gRNA[mu])
   N <- length(ind)
   # Converts into character matrix (not the most elegant
      way...)
   Y[[mu]] <- matrix(as.vector(melt(lapply(as.character(
      data[[2]][ind]),strsplit,split=""))$value),nrow=N,
      ncol=2*D,byrow=TRUE)
}

### Fit model for each gRNA

# Preallocate a list of model resuls
res <- vector('list',G)

# Loop through gRNAs, for each one fit a model and get the
    corrected read count
# This can be parallelised for speed
for(mu in 1:G){
   # Begin tryCatch (catches any errors instead of
      stopping the loop)
   tryCatch({
      # Indices for barcodes matched that the current gRNA
      ind <- which(data[[1]]==gRNA[mu])
      # Vector of read counts
```

```
        counts <- data[[3]][ind]
        # Fit the model
        res[[mu]] <- fit_model(Y[[mu]], counts)
    }, error=function(e) NULL) #End tryCatch
} # End loop over gRNAs
```

This analysis calls functions stored in the R scripts `fit_model.R`, `LL.R` and `hamming.R`. A csv file of counts per gRNA for each sample can then be exported.

4.7 Diagnostic plot: Number of UMI-corrected counts versus number of reads per gRNA

4.7.1 Calculating the number of reads per gRNA

To calculate the number of reads per gRNA for each sample, I wrote the script `reads_per_gRNA.py`. This takes a 3 column tab-separated inputfile with the following columns: [0] read ID [1] gRNA chr:start-stop [2] 14 nt barcode and outputs a csv file with two columns: [0] gRNA chr:start-stop , [1] number of reads. The script can be invoked as follows:

```
for i in *length14; do python ./reads_per_gRNA.py "$i";
   done
```

4.7.2 Wrapping gRNA counts of all samples into a table

While the Bayesian model ouputs a csv file containing the counts per gRNA for each sample directly, the output of the script `collapse-barcodes.py` (used to count gRNAs without error correction or to perform a naive PCR error correction) outputs one table per sample. This data can be merged into a single table listing for each gRNA the count in each sample using the script `make_table_from_counts.py`. This script also adds information about which gene is targeted by each gRNA. The script takes a variable number of inputfiles to wrap into a table:

```
python MakeTable_from_counts.py INPUTFILE1 INPUTFILE2 ...
   INPUTFILEn
```

This was used to generate the tables

4.7.3 Generating the plots of counts versus number of reads for each gRNA

The number of reads per gRNA were plotted against the counts per gRNA, derived either without error correction, with naive PCR error correction or Bayesian PCR error correction (as described above), using the script `plot_counts_vs_number_of_reads.py`.

4.8 Diagnostic plot: Counts versus number of sorted cells

This script accepts three user-arguments, a dataframe of gRNA counts (c), a dataframe of numbers of sorted cells per sample (n), and a samplesheet (s) and returns a scatterplot of sorted cells versus counts with one data point per sample in the samplesheet.

4.9 Enrichment analysis using DESeq2

4.9.1 Preparing Bayesian error correction output for DESeq2

DeSeq requires for each experiment a list of counts per gRNA.The data for each experiment were extracted from the Bayesian analysis output. gRNAs where 3/4 of the counts in a given experiment are 0 (or NA) are removed before enrichment analysis as follows:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import collections as coll
import re
import pylab

###Functions###

def remove_underscore_column_names(dataset):
    dataset_renamed = dataset.rename(columns=lambda x: re.
       sub(r"_.*", "",x))
    return dataset_renamed

def generate_samplefile(samplefilename):
    samplefile =[]
    with open(samplefilename) as infile:
        for line in infile: #remove everything after the
           first underscoe and remove newline
            samplefile.append(re.sub(r"_.*", "", str(line.
               rstrip("\n"))))
    return samplefile

def remove_3quarters_NAs(dataframe):
    df_with_NAs = dataframe.replace('0', np.nan)
    thresh = int(len(df_with_NAs.columns)/4*3)
    NA_removed =df_with_NAs.dropna(axis='index', thresh=
       thresh)
    return NA_removed
```

```
 def save_df_for_samplefile(samplefile, samplefilename): #
    this function calls all other functions
    df_samplefile = df_bayes[samplefile]
    df_samplefile = remove_3quarters_NAs(df_samplefile)
    df_samplefile = df_samplefile.fillna(0)
    df_samplefile.to_csv(str(samplefilename)+'_counts.csv
       ')

###Script###

df_bayes = pd.DataFrame.from_csv('path_2/
   bayesian_corrected_counts.csv', header=0, sep=',',
   index_col=0)

samplefilename = 'path_2/samplefile.txt'
samplefile =generate_samplefile(samplefilename)

save_df_for_samplefile(samplefile, samplefilename)
```

4.9.2 Visualising enrichment

4.9.3 Correlation of Log2Fold enrichment scores from DESeq2 between experiments

```
library ("DESeq2")

experiment_info <- read.table("path2/
   ThisExperiment_DeSeq2_ColData.csv", sep=",", header=
   TRUE, blank.lines.skip = TRUE, row.names = "Sample.name
   ")
counts<-read.table("path2/ThisExperiment_counts_bayes_new.
   csv", sep=",", header=TRUE, na.strings="NaN", check.
   names=FALSE, row.names=1)


counts_Batch5[is.na(counts_Batch5)] <- 0  #replace NAs
   with 0
dds_Batch5<- DESeqDataSetFromMatrix(countData=
   counts_Batch5, colData = experiment_info_Batch5, design
    = ~treatment)
#converting counts to integer mode
dds_Batch5<-DESeq(dds_Batch5)

res<-results(dds_Batch5)
resOrdered <- res[order(res$padj),]
resOrdered
```

Bibliography

[1] `http://hannonlab.cshl.edu/fastx_toolkit/`.

[2] `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`.

[3] `http://www.biostars.org/p/1195/`.

[4] `http://www.biostars.org/p/14614/`.

[5] `http://www.biostars.org/p/6219/`.

[6] Charles Berry, Sridhar Hannenhalli, Jeremy Leipzig, and Frederic D Bushman. Selection of Target Sites for Mobile DNA Integration in the Human Genome. *PLoS Comput Biol*, 2(11):e157, 2006.

[7] Bram De Craene and Geert Berx. Regulatory networks definingEMT during cancer initiation andprogression. *Nat Rev Cancer*, 13(2), 2013.

[8] W James Kent, Charles W Sugnet, Terrence S Furey, Krishna M Roskin, Tom H Pringle, Alan M Zahler, and David Haussler. The human genome browser at UCSC. *Genome Research*, 12(6):996–1006, June 2002.

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

[10] H Li, B Handsaker, A Wysoker, T Fennell, J Ruan, N Homer, G Marth, G Abecasis, R Durbin, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, August 2009.

[11] Prashant Mali, John Aach, P Benjamin Stranges, Kevin M Esvelt, Mark Moosburner, Sriram Kosuri, Luhan Yang, and George M Church. CAS9 transcriptional activators for target specificity screening and paired nickases for cooperative genome engineering. *Nat Biotechnol*, 31(9):833–838, August 2013.

[12] M Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet journal*, 2011.

[13] Youhei Namiki, Takashi Ishida, and Yutaka Akiyama. Acceleration of sequence clustering using longest common subsequence filtering. *BMC Bioinformatics*, 14(Suppl 8):S7, May 2013.

[14] H. Pages, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms.* R package version 2.30.0.

[15] Herve Pages. *BSgenome: Infrastructure for Biostrings-based genome data packages.* R package version 1.28.0.

[16] A R Quinlan and I M Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, March 2010.

[17] P Rice, I Longden, and A Bleasby. EMBOSS: the European Molecular Biology Open Software Suite. *Trends in Genetics*, 16(6):276–277, June 2000.