Guidescan2 Software Manual

Henri Schmidt[1]

[1]Department of Computer Science, Princeton University

August 17, 2022

Contents

1 Installation

1.1 Software Dependencies

The off-target enumeration tool, Guidescan2, relies on the installation of a modern C++ compiler and associated buildtools. Explicitly, the tool has the following dependencies.

- CMake version 3.1.0 or higher
- C++ compiler that supports C++11 features such as:
  - regex
  - constexpr
  - default constructors
- C++ support for POSIX threads (pthreads)

It is known that GCC version 4.9 or greater and clang 3.1 or greater with libc++ are both supported. But to error on the side of caution, the most up to date and stable version of GCC is recommended. As often high-performance computing clusters do not possess recent versions of C++ compilers, it is therefore recommended that the software is built and executed in some sort of virtual environment. We have had success with Singularity and Docker for these special cases, though if a modern compiler is available, that can be used instead.

The aforementioned dependencies are the bare minimum to obtain anything useful out of the tool. However, it is strongly recommended, and necessary in some our pipelines, that biological computing tools are also available. In particular, the Samtools suite is useful since the output database will be in a SAM file format [**danecek2021twelve**]. There are also bindings for the tool in several programming languages that may help users to interface with the Guidescan2 databases programatically.

Helper scripts for various pipelines require a Python3 and Conda installation. We include a Conda environment file that contains all the necessary dependencies, though the scripts should run fine if standard data science packages are installed.

1.2 Build Procedure

Building the off-target enumeration tool, Guidescan2, is straightforward usign CMake once the dependencies have been installed. First set the working directory to the guidescan-cli project root. Then execute the following commands.

```
$ mkdir build; cd build
$ cmake -DCMAKE_BUILD_TYPE=RELEASE ..
$ make
```

The executable will be output in the `build/bin/` directory under the name `guidescan`. To make things convenient, we recommend that you add the program to your path. One way to do this is to place the executable under `$HOME/bin/` and then append this directory to your `$PATH`. The following code snippet does this and then permanently updates your `$PATH` in your bashrc.

```
$ mkdir -p $HOME/bin
$ cp bin/guidescan $HOME/bin
$ echo export PATH=$PATH:$HOME/bin >> ~/.bashrc
$ source ~/.bashrc
```

2 Command Line Interface

For ease of exposition, we will assume Guidescan2 is installed under your path and can be executed by simply running the command `guidescan`. However, everything will apply even if you execute Guidescan2 using its absolute path.

To run the Guidescan2 off-target enumeration and view its sub-commands, simply execute the program with the `--help` flag.

```
$ guidescan --help
```

There are three sub-commands for the Guidescan2 tool.

- `index`: Constructs a genomic index from a FASTA file.
- `enumerate`: Enumerates off-targets against a reference genomic index for a particular set of kmers.
- `http-server`: Spawns a local http-server that responds to off-target queries. That is, for a GET request containing a sequence of interest, enumerates all off-targets for that sequence.

The sub-commands each have their own interfaces, accessed as such:

```
$ guidescan [SUBCOMMAND] --help
```

For almost all use cases, it will suffice to stick to the main `enumerate` and `index` commands, the others are only helpers for very specific purposes.

2.1 Genome Indexing

```
$ guidescan index --help
```

The `index` sub-command takes a FASTA file as input and constructs a compressed genomic index that is used for off-target search. The index is stored as three files in the current working directory with the name `*.index.*`. The index consists of a forward and reverse strand index as well as a small metadata file containing chromosome information. Auxiliary files with extensions `.forward.fna` and `.reverse.fna` are written to disk during the indexing process but these can be deleted once the genome is constructed.

The `index` step is the only step that requires a decent amount of memory to execute. For the human reference genome, 32GB of memory will suffice. However once the indices are constructed they can be transferred across devices as they are relatively small. Indices for a wide variety of genomes are available at ????. And though this steps requires a moderate amount of memory, it is quick to execute, taking under thirty minutes on a Lenovo Thinkpad t490 with 32GB of RAM and an 8-core processor.

2.2 Off-Target Enumeration

```
$ guidescan enumerate --help
```

Once an index is constructed, off-target enumeration proceeds in an online fashion with the `enumerate` sub-command. Input to this command is a pointer to a reference index and a set of kmers to evaluate off-targets against. PAMs are included in this kmer set and are matched at the specified end of the kmer. The output is a database in SAM file format containing kmers that have passed the Guidescan2 filtering step. Complete off-target information is hex-encoded in the `of` field and can be decoded using an included Python3 script described later.

The command understands several options. Most importantly is the flag `-k/--mismatches` that describes the mismatch radius to search for kmer matches within. Typically, this parameter is set to a small value of, typically at most 3 or 4, as the search complexity grows exponentially with this parameter. Second, it is often useful to specify an alternative set of PAMs to match off-targets against since different CRISPR systems have varying PAM specificity. Of course this step can be emulated using repeated calls to `enumerate` with different PAMs, but we include this as a feature for convenience. Multi-threading across cores is available with the `--threads` option. This may result in out of order entries compared to the input kmer set since threads may complete off-target enumeration at different speeds.

We should also note that kmers that have multiple perfect matches are thrown away by default. This behavior can be turned off...

The enumerate command **requires** a set of genomic sequences to evaluate, we refer to these as the *kmer set*. The kmer set is specified in a very flexible CSV format described below. It can be automatically generated by the tool or manually specified. The kmer set consists of six columns:

```
id,sequence,pam,chromosome,position,sense
```

of which only `id`, `sequence` and `sense` are required, though all must be specified in the header. An example is shown in (Figure 1). Notice that for the third example, we don't specify a PAM. In this case, we simply match off-targets against the kmer with no regards to PAM specificity — off-targets don't need a PAM.

```
id,sequence,pam,chromosome,position,sense
ce11_example_1,GCCGTTCAGGAGCTCGACGA,NGG,chrIV,5499,-
ce11_example_2,CAAAATATGAAATTTTCAAG,NGG,chrIV,23896,+
ce11_example_3,TCTACTGAAAGTTTGCAAAA,,chrIV,5499,-
ce11_example_4,TATAAACTGTCAAAGTTGAG,NGG,chrIV,23703,+
```

Figure 1: : An example kmer file for the Caenorhabditis elegans genome.

2.3 Kmer Set Generation

As mentioned earlier, *kmer set* generation can be peformed manually, for example, when you want to evaluate a specific set of gRNAs. Additionally, Guidescan2 can also generate *kmer set* automatically. To generate *kmer sets* we include a Python3 script, `generate_kmers.py`, that generates a set of kmers against an input FASTA file. To use the script simply write

```
$ python generate_kmers.py FASTA_FILE
```

and the output CSV will be sent directly to `stdout`.

The script understands several options, though it is only mandatory to specify the FASTA file with which kmers are scanned. Additionally, one can specify PAM and kmer length, with defaults set to NGG and 20. By default kmers are only generated from chromosomes that are sufficiently long, but this can be specified using the `--min-chr-length` option. It also uses positional information as an identifier, but an additional prefix can be specified using the `--prefix` option.

The dependencies for the script are Python3 and the Biopython toolkit.

3 Off-Target Databases

Guidescan2 outputs databases in the SAM file format, a common file format used in Bioinformatics. At minimum, the Guidescan2 database contains the off-target set for each screened kmer as well as chromosomal structure information. Importantly, **the header is essential** for downstream processing; take care to not delete it accidently. The additional fields position, chromosome, and strand are filled in only if they are specified in the *kmer set* for the corresponding kmer.

Though convenient for storage, compression, and a variety of bioinformatics tasks, the off-target information is difficult to pull out. Namely, off-target information is hex-encoded in the attributes

field with label `of` and is not human-readable. As such, we include a script to decode the off-target information into human-readable format.

3.1 Decoding

The script `decode_database.py` is a Python3 script that decodes Guidescan2 databases sequentially into a human-readable CSV format. It takes the Guidescan2 SAM/BAM database and the original FASTA file as input and outputs a CSV to `stdout`. The script outputs information in two modes, denoted *succinct* and *complete*, which we will describe below.

For most use cases *succinct* mode will be sufficient. In this mode each kmer's off-targets are decoded and the information is summarized into multiple columns. Namely, we ouput the counts for matches at various distances and an overall score denoted specificity which is defined in the Supplementary Text. We note that the specificity is only defined for 20-mers with NGG PAMs; this column of the output will be empty otherwise. We also include additional metadata with obvious meaning upon examination.

In *complete* mode we explicitly write out the sequence and genomic location of all off-targets for each kmer. In addition, in the 20-mer case we include the CFD score from which our specificity score is derived. We warn that since some kmers can have up to hundreds of thousands of off-targets, *complete* mode can result in an extremely large output. If this is of concern we recommend that you re-run Guidescan2 with a smaller value of mismatch parameter and then decode the databases again.

To run the script simply write,

```
$ python scripts/decode_database.py GUIDESCAN_SAM_DB FASTA_FILE
```

The script **depends** on the two pickled files in the subdirectory `scripts/cfd/`. As such, it is required that the script and the sub-directory `scripts/cfd/` are in the location for the program to properly execute. By default this is the case, but we caution that when moving the script, move the `scripts/cfd/` sub-directory as well.

4 Guidescan2 Pipelines

4.1 Analysis of Individual gRNAs

Here we introduce a pipeline for the analysis of a small set of gRNAs that have been gathered from external sources. In our example, we will analyze a subset of genes targeted in a CRISPRko essentiality screen [**Wang1096**]. The files for this example can be found in the aforementioned publication, but are included as CSV files in the `examples/` sub-directory. The two files we will use contain the set of gRNAs used in the knockout screen (`sabatini_grnas.csv`) and gRNA abundance after harvesting at initial and final time points (`sabatini_read_counts.csv`).

```
#Chromosome        Accession.version
1         NC_000001.11
2         NC_000002.12
3         NC_000003.12
...
X         NC_000023.11
Y         NC_000024.10
```

Figure 2: chr2acc file that maps standard chromosome names to NCBI accession identifiers. Guidescan2 uses these identifiers internally since they uniquely identify organism version and chromosome simultaneously.

To start we will index the reference genome to analyze off-targets against. Since the Sabatini2015 screen is performed in human cells, the human reference genome *hg38* is a natural choice. We will use the RefSeq version of this reference genome which is provided by the National Center for Biotechnology Information at the address:

`https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26/`

Once downloaded the reference genome must be decompressed into a raw FASTA file. To do this one can run the following sequence of commands, assuming the downloaded tarball has name `hg38.tar`.

```
$ tar -xvf hg38.tar
$ cd ncbi-genomes-2022-tarball
$ gunzip GCF_000001405.26_GRCh38_genomic.fna.gz
```

It will also be useful to include the file `chr2acc` which maps chromosome names to NCBI accessions, which are used internally in Guidescan2. For example, this file can be found on the NCBI FTP server at the location:

`.../Primary_Assembly/assembled_chromosomes/chr2acc`

and it takes the form in (Figure 2).

Now, since the reference genome is a raw FASTA file, we can run Guidescan2 indexing on it as follows. Note that this step can require up to 32GB of memory for large genomes. When generating the index for *hg38*, this step took a maximum of 30GB of memory. Large temporary files with extension `.sdsl` will be created and subsequently deleted during this process.

```
$ guidescan index --index hg38 GCF_000001405.26_GRCh38_genomic.fna
$ rm GCF_000001405.26_GRCh38_genomic.fna.*.dna
```

Note that we specify the prefix to ensure the commands are short and delete the temporary files `*.dna` once the index has been constructed. If memory constraints are an issue, we remind the user that pre-constructed indices can be found here.

With the index constructed, we then turn to processing the gRNAs into the *kmer set* format for processing by Guidescan2. For the sake of example, we will randomly select of 250 genes to process.

```
$ awk -F, '(NR > 1) { print $1 }' examples/sabatini_grnas.csv > id_list.txt
$ cat id_list.txt | sed -e '/CTRL.*/d' | sed -r 's/sg(.*)_.*/\1/' > gene_list.txt
$ uniq gene_list.txt | shuf | head -n 250 > random_genes.txt
$ cat random_genes.txt | sed -r 's/.*/sg\0_/' > random_gene_ids.txt
```

The preceding code grabs the `sgRNA ID` field of the CSV, drops all control gRNAs, parses the gene, removes duplicates, and then finally selects a random subset of genes. We include this as an illustrative example of the custom processing that may need to performed in order to analyze your gRNA set; it differs from dataset to dataset.

With all the genes we want to analyze selected, we finally construct our *kmers set*. We do this by selecting all gRNAs in our gene set, sorting the rows, appending a PAM column, and adding in our header. We also delete the temporary files that are no longer needed.

```
$ grep -f random_gene_ids.txt examples/sabatini_grnas.csv > grnas.csv
$ awk -F, '{ print $1 "," $6 ",NGG," $3","$4","$5 }' grnas.csv > kmers.csv
$ sed -i '1i id,sequence,pam,chromosome,position,sense' kmers.csv
$ rm id_list.txt gene_list.txt random_genes.txt random_gene_ids.txt grnas.csv
```

At this point, it is convenient (though not necessary) to update the chromosome names with NCBI accessions. As an example, we include the following Python script `examples/chr2acc.py` that will perform this update (Figure 3). We do not include a general solution to this problem since it depends on the format of the gRNA given to analyze, but for the example simply execute the following.

```
$ python examples/chr2acc.py chr2acc.txt kmers.csv > kmers2.csv
$ mv kmers2.csv kmers.csv
```

Finally, our *kmer set* is ready to run through Guidescan2. After moving everything to the correct directory, we can evaluate our *kmer set* with the following command.

```
$ guidescan enumerate ~/ncbi-*/hg38 -f kmers.csv -n 1 -o db.sam -a NAG
```

As an estimate of time, this process took less than 10 minutes on a Thinkpad t490. Since kmers are evaluated synchronously and written in real time, progress can be measured via the number of

```
import sys

if len(sys.argv) < 3:
    print("usage: python chr2acc.py [chr2acc.txt] [kmers.csv]")
    sys.exit(1)

chr2acc = {}
with open(sys.argv[1]) as f:
    next(f)
    for l in f:
        words = l.split()
        chr2acc[words[0]] = words[1]

with open(sys.argv[2]) as f:
    print(next(f), end='')
    for l in f:
        words = l.split(',')
        words[3] = chr2acc[words[3][3:]] # strips 'chr' prefix
        print(','.join(words), end='')
```

Figure 3: An example script for mapping chromosome to accession names.

lines in the output database. That is,

```
$ wc -l db.sam
```

tells you how many kmers have successfully been processed by Guidescan2.

To make our output human readable, we run our decoding script on the output database, passing in the reference FASTA file as input.

```
$ python scripts/decode_database.py db.sam ~/ncbi-*/*.fna > processed.csv
```

Finally, we are done! To enable the analysis of large sets of kmers, say on the order of 10^6 kmers, we recommend either reducing the mismatch to 2 or parallelizing across several compute nodes. The following sections on off-target database construction describe simple strategies for parallelization.

4.2 Off-Target Database Construction

Here we describe a program to construct genome-wide off-target databases for organisms. To ensure this example runs quickly, we will construct a databse for the C. elegans genome. An in particular, we will use the RefSeq version provided by the National Center for Biotechnology Information at

the address:

`https://www.ncbi.nlm.nih.gov/assembly/GCF_000002985.6/`

Once downloaded the reference genome must be decompressed into a raw FASTA file. To do this one can run the following sequence of commands, assuming the downloaded tarball has name `cell.tar`.

```
$ tar -xvf cell.tar
$ cd ncbi-genomes-2022-tarball
$ gunzip GCF_000002985.6_WBcel235_genomic.fna.gz
```

Now, since the reference genome is a raw FASTA file, we can run Guidescan2 indexing on it as follows. Note that this step can require up to 32GB of memory for large genomes. When generating the index for *hg38*, this step took a maximum of 30GB of memory. Large temporary files with extension `.sdsl` will be created and subsequently deleted during this process.

```
$ guidescan index --index cell.index GCF_000002985.6_WBcel235_genomic.fna
$ rm GCF_000002985.6_WBcel235_genomic.fna.*.dna
```

Note that we specify the prefix to ensure the commands are short and delete the temporary files `*.dna` once the index has been constructed. If memory constraints are an issue, we remind the user that pre-constructed indices can be found here.

At this point, we will deviate from the analysis in the previous section. Instead of looking at a targeted set of gRNAs received from a third-party, we will construct a genome wide set of gRNAs using Guidescan2. To do this we need to specify the *targetable space* of the genome. For standard CRISPR Cas9 enzymes, this corresponds to the set of all 20-mers followed by NGG PAMs; but it varies across CRISPR systems. Accordingly, we run the `generate_kmers.py` script with our desired parameters.

```
$ python scripts/generate_kmers.py --pam NGG --prefix "example-"\
  cell.fna > cell.kmers
```

To enable parallel execution, we partition our kmer set across 100 pieces. The *nix `split` utility is useful for this purpose. And then we append the header to each split. Since this is a little bit more complex, we include it as a script (Figure 4). With the kmers set split into pieces, we can run Guidescan2 seperately on each piece and then merge the resulting SAM files together. The exact steps necessary depend upon on the computing resources available, but it should be straightforward. As an example, we will enumerate gRNAs in a sequential fashion.

```
$ for f in cell.*.csv; do\
```

```
  guidescan enumerate ce11.index -f $f -n 1 -o $f.sam -a NAG;\
  done
```

Note that this step would take a **very** long time to complete if run sequentially. For the example here, it would take around 1,000 minutes running on a single core on a Thinkpad t490; the time would increase exponentially for organisms with longer genomes. Assuming this step is completed, we finish database construction by merging the databases together. To do this, we strip the headers from all but one of the files and concatenate them together. This can be done using samtools, but here we will just use `sed` and `cat`.

```
  $ cp cell.000.sam cell.sam
  $ sed -E "/@.*/d" cell.*{1..99}.csv.sam >> cell.sam
```

And at this step, database generation is complete. For storage, we recommend compressing the file to a BAM format using Samtools. To make our output human readable, we can run our decoding script on the output database, or subsets thereof, passing in the reference FASTA file as input. Remember to keep the header when splitting the database into subsets. It can also be convenient to perform this step on the individual database splits in parallel, though this step is very quick.

```
  $ python scripts/decode_database.py cell.sam\
↪   GCF_000002985.6_WBcel235_genomic.fna > ce11.csv
```

```
#!/bin/bash

split -d -n l/100 -a 3 --additional-suffix .csv cell.kmers cell.
for f in cell.*.csv; do
    awk 'BEGIN {print "id,sequence,pam,chromosome,position,sense"} {print $1}'\
        $f > $f.header
    mv $f.header $f
done

# first file contains two headers now; we remove
tail -n +2 cell.000.csv > tmp.csv
mv tmp.csv cell.000.csv
```

Figure 4: Bash script to split `cell.kmers` file into 100 partitions for parallelization.

4.3 Off-Target Database Construction for F1-Cross Genomes

To construct databases for hybrid genomes and enable allele specific targeting, we need reference sequences for both alleles. In our case of generating allele specific guides for an F1 hybrid B6/129S1

mouse genome, we used the *mm38* reference genome and a synthetic *129S1* genome. If high quality genomes and mappings between them are available for both alleles, then it is not necessary to generate a synthetic genome. But this was not the case for 129S1.

We co-opted the MMARGE tool built for epigenetic data to construct both a *129S1* reference genome and a mapping between *129S1* and *mm38* [**link2018mmarge**]. We used the reference assembly `GRCm38_68.fa` from Ensemble release 68 and the sequence variation files from the Mouse Genome Project [**keane2011mouse**]. At the time of publication, the VCF files are available for installation at:

`ftp://ftp-mouse.sanger.ac.uk`

Essentially, we pulled down the set of indels/SNPs as a VCF file and processed them for usage with MMARGE. We also split the genome into chromosomes, since this is required for processing. We refer the reader to [**link2018mmarge**] for full details on constructing synthentic genomes with MMARGE since it is rather detailed. However, we include the code we used for this in (Listing 1).

At this point, we assume the reader has two reference genomes, referred to as `grcm38.fa` and `129s1.fa` and a mapping between the coordinates of the two. We now run Guidescan2 off-target database construction as in sub-section 4.2 for each of these genomes seperately. This will result in two Guidescan2 databases `grcm38.sam` and `129s1.sam` for the corresponding references. These consist of the sets of gRNAs that have passed standard Guidescan2 filtering. From these databases, we construct two *kmer sets* and then run these against the indices for the opposite allele with a an exact match threshold of 0. This will result in two sets of gRNAs that have exact matches on only one allele, as well as complete off-target information for both alleles.

```
DIR=inputs/
MMARGE=MMARGE.pl
SNPS=$DIR/129S1_SvImJ.mgp.v5.snps.dbSNP142.vcf
INDELS=$DIR/129S1_SvImJ.mgp.v5.indels.dbSNP142.normed.vcf

mkdir -p $DIR

if [ ! -f $SNPS ]; then
    echo "Downloading and pre-processing latest SNP files for $(basename $SNPS)."
    echo "========================================================================="
    wget
    ↪  ftp://ftp-mouse.sanger.ac.uk/REL-1505-SNPs_Indels/strain_specific_vcfs/$(basename
    ↪  "$SNPS").gz -O "$SNPS".gz
    gunzip "$SNPS".gz
    grep "^#" $SNPS > "$SNPS".sorted
    grep -v "^#" $SNPS | sort -k1,1 -k2,2n --parallel=24 >> "$SNPS".sorted
fi

if [ ! -f $INDELS ]; then
    echo "Downloading and pre-processing latest INDEL files for $(basename
    ↪  $INDELS)."
    echo "========================================================================="
    wget
    ↪  ftp://ftp-mouse.sanger.ac.uk/REL-1505-SNPs_Indels/strain_specific_vcfs/$(basename
    ↪  "$INDELS").gz -O "$INDELS".gz
    gunzip "$INDELS".gz
    grep "^#" $INDELS > "$INDELS".sorted
    grep -v "^#" $INDELS | sort -k1,1 -k2,2n --parallel=24 >> "$INDELS".sorted
fi

REF_FTP=$(head $INDELS | PERL_BADLANG=0 perl -Xne 'print $1 if
↪  /reference=(ftp.*)/')
REF=$DIR/$(basename $REF_FTP)
if [ ! -f $REF ]; then
    echo "Installing reference genome from: $REF_FTP"
    echo "========================================================================="
    wget "$REF_FTP".gz -O $REF.gz
    gunzip $REF.gz
fi
```

```
echo "Splitting genome into chromosomes."
echo "================================================================================"

if [ ! -f scripts/faSplit ]; then
    wget http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64.v385/faSplit
    chmod u+x scripts/faSplit
fi

mkdir -p genomes
./scripts/faSplit byname $REF genomes/

for f in genomes/*; do
    mv $f genomes/chr$(basename $f)
done

echo "Preparing files with MMARGE"
echo "================================================================================"

$MMARGE prepare_files -core 8 -files "$SNPS".sorted "$INDELS.sorted" -genome
↪  genomes -dir . -genome_dir .
```

Listing 1: Example code to construct to synthetic *129S1* genome using MMARGE, including pre-processing steps.

5 Bibliography