AllMethods

May 4, 2022

Preparations

Compile the C++ module for the python script The python scripts (within `./tools/`) use the FPA algorithm through a C++ module. That has been compiled for python3.6 on x86_64 Linux. The compilation requires the pybind11 libraries (https://pybind11.readthedocs.io/en/stable/). An example of compile command is in `tools/fpa/compile.txt`:

[ ]:
```
c++ -O3 -Wall -shared -std=c++11 -fPIC -I/usr/include/python3.6m -I/usr/local/include/python3.6 \
    -I$SOME/$DIR/pybind11/include fpa.cpp -o fpa_ext2.cpython-36m-x86_64-linux-gnu.so
```

Compile the modified SvABA The code has been tested and compiled on Ubuntu 18.04.5 LTS with gcc version 7.5.0. It may work on other systems after re-compilation. That can be performed with the following commands:

[ ]:
```
cd svaba
./configure
make

cp ./src/svaba/svaba ../tools
cd ..
```

Get Ebert et al. data The **freeze3** of Ebert et al. data were downloaded and the variant calls were combined. The correct reference genome was downaloaded and split by chromosomes. One of the python scripts reads the data as genotypes and the format has to be converted (the other one does the conversion internally).

[ ]:
```
mkdir data_sv

URL1kG=http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/HGSVC2

wget $URL1kG/release/v1.0/integrated_callset/freeze3.snv.alt.vcf.gz -P data_sv
wget $URL1kG/release/v1.0/integrated_callset/freeze3.indel.alt.vcf.gz -P data_sv
wget $URL1kG/release/v1.0/integrated_callset/freeze3.sv.alt.vcf.gz -P data_sv

cat <( zcat data/freeze3.indel.alt.vcf.gz ) \
    <( zcat data/freeze3.snv.alt.chroms.vcf.gz | tail -n +211 ) \
    <( zcat data/freeze3.sv.alt.chroms.vcf.gz | tail -n +211 ) | bgzip -c > data_sv/tmp.vcf.gz
bcftools sort -Oz -o data_sv/freeze3.snv_indel_sv.alt.vcf.gz -T temp data_sv/tmp.vcf.gz

mkdir ref
wget $URL1kG/technical/reference/20200513_hg38_NoALT/hg38.no_alt.fa.gz -P ref

samtools faidx ref/hg38.no_alt.fa.gz
for i in `seq 1 22` X Y; do
 samtools faidx ref/hg38.no_alt.fa.gz chr$i > ref/hg38.no_alt.chr$i.fa
done

for i in `seq 1 22` X Y; do
 bcftools view -r chr$i data_sv/freeze3.snv_indel_sv.alt.vcf.gz | \
  bcftools query -f '%CHROM\t%POS\t%REF\t%ALT[\t%GT]\t%INFO/VARTYPE\n' > data_sv/freeze3.snv_indel_sv.gt2.
 ↪chr$i.txt &
done
```

Get Platinum and Chromium data Another reference genome was downloaded for the short reads. The Chromium data were downloaded as bams and vcf. The Platinum data were downloded as crams and vcf; the former were converted to bams. GRCh37 was used for the validation of the loci with the children data (not shown in detail as the data need authorized access; methods are highly similar).

[ ]:
```
URL1kG=http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/GRCh38_reference_genome

wget $URL1kG/GRCh38_full_analysis_set_plus_decoy_hla.fa -P ref
wget $URL1kG/GRCh38_full_analysis_set_plus_decoy_hla.fa.fai -P ref

mkdir data_plat data_chrom

URL1kG=http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/illumina_platinum_pedigree/data/CEU/
 ↪NA12878

wget $URL1kG/alignment/NA12878.alt_bwamem_GRCh38DH.20150706.CEU.illumina_platinum_ped.cram -P data_plat
wget $URL1kG/alignment/NA12878.alt_bwamem_GRCh38DH.20150706.CEU.illumina_platinum_ped.cram.crai -P␣
 ↪data_plat

URLS3=https://s3.eu-central-1.amazonaws.com/platinum-genomes/2017-1.0/hg38
wget $URLS3/small_variants/NA12878/NA12878.vcf.gz -P data_plat
wget $URLS3/small_variants/NA12878/NA12878.vcf.gz.tbi -P data_plat

samtools view -b -T ref/GRCh38_full_analysis_set_plus_decoy_hla.fa -o data_plat/NA12878.bam \
              data_plat/NA12878.alt_bwamem_GRCh38DH.20150706.CEU.illumina_platinum_ped.cram

URLGIAB=https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878

wget $URLGIAB/10Xgenomics_ChromiumGenome_LongRanger2.0_06202016/NA12878_GRCh38.bam -P data_chrom
wget $URLGIAB/10Xgenomics_ChromiumGenome_LongRanger2.0_06202016/NA12878_GRCh38.vcf.gz -P data_chrom
wget $URLGIAB/10Xgenomics_ChromiumGenome_LongRanger2.0_06202016/NA12878_GRCh38.vcf.gz.tbi -P data_chrom
```

Get gnomAD and Sasani et al. data

[ ]:
```
mkdir data_dnm

for i in `seq 1 10`; do
 wget https://gnomad-public-us-east-1.s3.amazonaws.com/release/2.1/mnv/genome/gnomad_mnv_genome_d$i.tsv.
 ↪bgz -P data_dnm
 zcat data_dnm/gnomad_mnv_genome_d$i.tsv.bgz | grep -v locus | awk '$7/$8>=0.9&&$7/$8<=1.1{print $5;print␣
 ↪$6}' \
 | sed 's/-/\t/g' > data_dnm/gnomad_mnv_genome_filt_d$i.tsv
done

cat data_dnm/gnomad_mnv_genome_filt_d*tsv | sort -V -k1,2 | uniq > data_dnm/gnomad_mnv_genome_filt_dAll.
 ↪tsv

cat data_dnm/gnomad_mnv_genome_filt_dAll.tsv | \
 awk 'BEGIN{nm=0;lim=2}{
 if($1==pc && $2-pp<10){ lst=lst"\n"$0;nm+=1}else{ if(nm>lim){print lst}nm=1;lst=$0} pc=$1;pp=$2}' \
 > data_dnm/gnomad_mnv_genome_filt_clust3.tsv

wget https://raw.githubusercontent.com/elifesciences-publications/ceph-dnm-manuscript/master/data/
 ↪second_gen.dnms.txt -P data_dnm
wget https://raw.githubusercontent.com/elifesciences-publications/ceph-dnm-manuscript/master/data/
 ↪third_gen.dnms.txt -P data_dnm

grep -v chrom data_dnm/third_gen.dnms.txt | cut -f 1,2,6,7 | sort -V > data_dnm/third_gen.dnms.tsv

wget http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/human_g1k_v37.fasta.gz -P ref
```

```
gunzip ref/human_g1k_v37.fasta.gz
samtools faidx gunzip ref/human_g1k_v37.fasta
```

Analysis of SV data

Identify TSMs Given the data, the script `tools/tsm_scan_SV2.py` does everything from the TSM scan to visualisation of the hits. The first command loops over the full data (all chroms, all individuals) while the second does the two haplotypes of NA12878 only. (The second one now only does chr1.)

[ ]:
```
mkdir results_sv

for i in `seq 1 22` X Y; do
 python3 tools/tsm_scan_SV2.py -c chr$i -f ref/hg38.no_alt.chr$i.fa -g data_sv/freeze3.snv_indel_sv.gt2.
 ↪chr$i.txt -t 500 \
     > results_sv/scan_out.chr$i.txt &
done

#for i in `seq 1 22` X Y; do
for i in 1; do
 for j in 16 17; do
  python3 tools/tsm_scan_SV2.py -c chr$i -f ref/hg38.no_alt.chr$i.fa -g data_sv/freeze3.snv_indel_sv.gt2.
 ↪chr$i.txt -t 500 -i $j
 done > results_sv/NA12878.scan_out.chr$i.txt
done
```

Filter and visualise TSM hits The output of the TSM scan is somewhat complex and follows the format of the original FPA tools (https://github.com/ariloytynoja/fpa). The meaning of the comma-separated fields is explained in `tools/tsm_scan_SV2.py`. Below, the first command selects the TSM hits in chr1 that are at least 25 nuc in length and prints the coordinates in a format comatible for the next step (see below 'Process the identified TSM loci ...'). The second command prints those hits on the screen. Note that it is necessary to index the hits by the coordinate (i.e. read everything in first) as this merges the possible duplicates created by near-by mutation clusters finding the same TSM locus.

[ ]:
```
grep -h -A1 -e "unmasked" results_sv/NA12878.scan_out.chr1.txt | grep -v "^--$" | paste -d " "  - - | \
 awk '{h[$3"."$4"."$5]=$12}END{for(i in h){split(h[i],r,",");l=r[8]-r[9]+1;if(l>19){print i}}}' | \
  while read a; do grep $a results_sv/NA12878.scan_out.chr1.txt | head -1 | awk '{OFS="\t"; print␣
 ↪$3,$4,$5,$2-15}'; done | \
   sort -V > results_sv/NA12878_regs25_chr1.bed

grep -h -A1 -e "unmasked" results_sv/NA12878.scan_out.chr1.txt | grep -v "^--$" | paste -d " "  - - | \
 awk '{h[$3"."$4"."$5]=$12}END{for(i in h){split(h[i],r,",");l=r[8]-r[9]+1;if(l>19){print i}}}' | \
  while read a; do grep -A19 $a results_sv/NA12878.scan_out.chr1.txt; done
```

Process the identified TSM loci and genotype them with short-read data The script `tools/tsm_alleles2.py` creates the two alleles for the listed loci, extracts and maps the short reads against these alleles, and finally computes the mapping depth in and around the differeing parts. Below, the first command creates (-c) alternative alleles for the given loci, then aligns (-a) the short reads to these, and finally scores (-s) the resulting alignments by the coverage. The second command uses the existing haplotypes (references for the mapping) and only aligns (-a) the short reads and scores (-s) the resulting alignments.

The output is: chrom, start, stop, depth upstream REF, depth inside REF, depth downstream REF, depth upstream ALT, depth inside ALT, depth downstream ALT

[ ]:
```
mkdir -p results_sv/NA12878/chrom results_sv/NA12878/plat results_sv/NA12878/alleles

python3 tools/tsm_alleles2.py -c -a -s -v data_sv/freeze3.snv_indel_sv.alt.vcf.gz -l results_sv/
 ↪NA12878_regs25_chr1.bed \
                            -d results_sv/NA12878/alleles -i NA12878 -f ref/hg38.no_alt.chr1.fa -r␣
 ↪data_chrom/NA12878_GRCh38.bam \
                            -b results_sv/NA12878/chrom \
```

```
  > results_sv/NA12878/chrom/results.txt

python3 tools/tsm_alleles2.py -a -s -d results_sv/NA12878/alleles -r data_plat/NA12878.bam -b results_sv/
 ↪NA12878/plat \
  > results_sv/NA12878/plat/results.txt
```

Determine the genotypes The genotypes were determined using a function coded in R. The function computes the probability of the data under different genotypes (allowing for 1% error), and then selects the genotype with the highest posterior probability if that is over 0.99. The minimum coverage for either of the alleles was 10 reads.

[5]:
```
score <- function(dat){
    e <- 0.01
    pr <- (1-e)^dat$dir*e^dat$dia
    ph <- 0.5^dat$dir*0.5^dat$dia
    pa <- e^dat$dir*(1-e)^dat$dia
    ps <- rowSums(cbind(pr,ph,pa))
    res <- cbind(pr/ps,ph/ps,pa/ps)
    mv <- apply(res,1,function(m){which(m==max(m)& m > 0.99)})
    mv[is.na(mv<0)] <- 0
    unlist(mv)
}

filter <- function(dat,mcov){
    (dat$dur>mcov & dat$dir>mcov & dat$ddr>mcov)|(dat$dua>mcov & dat$dia>mcov & dat$dda>mcov)
}

dat.plat <- read.table("results_sv/NA12878/plat/results.txt")
colnames(dat.plat)  <-  c("chrom","start","stop","dur","dir","ddr","dua","dia","dda")

dat.chrom <- read.table("results_sv/NA12878/chrom/results.txt")
colnames(dat.chrom)  <-  c("chrom","start","stop","dur","dir","ddr","dua","dia","dda")

keep.plat <- filter(dat.plat,9)
keep.chrom <- filter(dat.chrom,9)
keep <- keep.plat & keep.chrom

gt.plat <- factor(score(dat.plat[keep,]),levels=c(0,1,2,3),labels=c("NA","RR","RA","AA"))
gt.chrom <- factor(score(dat.chrom[keep,]),levels=c(0,1,2,3),labels=c("NA","RR","RA","AA"))

cbind.data.frame(dat.plat[keep,1:3],gt.plat,gt.chrom)
```

A data.frame: 4 ÃŮ 5

chrom <fct>	start <int>	stop <int>	gt.plat <fct>	gt.chrom <fct>
chr1	42904851	42904857	AA	RA
chr1	43911614	43911634	AA	AA
chr1	201549678	201549683	AA	AA
chr1	219336031	219336032	AA	AA

Analysis of short-read data

Identify repeats and low-complexity sequence To reduce erroneous signal from incorrect mapping and spurious mutations, repeat elements and low-complexity sequence were identified using RepeatMasker (v.4.0.5) and dustmasker (v.1.0.0); the detected regions were processed with bedtools (v.2.26.0). These regions are ignored at the later stage of the analysis.

[ ]:
```
REF=ref/GRCh38_full_analysis_set_plus_decoy_hla.fa

mkdir ref/RM

repeatmasker -xsmall -gff -dir ref/RM -pa 20 -species "human" $REF
```

```
dustmasker -in ref/RM/$REF.masked -outfmt acclist | sed 's/>//' > ref/RM/$REF.dust.bed

grep -w ^"chr[0-9]*" ref/RM/$REF.dust.bed | \
  awk -F"\t" '{match($1,"chr[0-9]*");OFS="\t";print substr($1,RSTART, RLENGTH),$2,$3}' > ref/RM/dust.bed

grep -w ^"chr[0-9]*" ref/RM/$REF.out.gff | \
  awk -F"\t" '{OFS="\t";print $1,$4,$5}' > ref/RM/repeats.bed

cat ref/RM/repeats.bed ref/RM/dust.bed | bedtools sort -i - | bedtools merge -i - > ref/repeats_dust.bed
```

Identify candidate loci using soft-clip and variant information Abnormalities were searched in the NA12878 Chromium and Platinum data. Soft-clipped but otherwise perfectly matching sequencing reads (SUGAR strings containing one M and one or two S's) were identified and clipping positions were recorded. These step very done with awk-scripts `tools/findClips.awk` and `tools/countClips.awk` and some bash. Clusters of variants were identified using bcftools and the awk-script `tools/findClusters.awk`. Based on the information of variants and soft-clip positions, clusters of more than one base mismatch or more than ten clipped reads within a 20-bp interval were identified as candidate loci. Candidate loci intersecting with identified repeat elements and low-complexity sequence were removed.

[ ]:
```
mkdir data_plat/clips data_plat/beds

# Originally "GRCh38_full_analysis_set_plus_decoy_hla" was used.
# For simplicity, here "hg38.no_alt" is used (may affect) and only chr1 is analysed.
#
#FAI=ref/GRCh38_full_analysis_set_plus_decoy_hla.fa.fai
FAI=ref/hg38.no_alt.chr1.fa.fai
BAM=data_plat/NA12878.bam

#grep -w chr[0-9]* $FAI | while read -r CHR LEN x y z; do
grep -w chr1 $FAI | while read -r CHR LEN x y z; do
  RES=data_plat/clips/NA12878_clips_$CHR.bed
  > $RES
  for pos1 in `seq 1 100000 $LEN`; do
    pos2=$(( $pos1 + 101000 ))
    samtools view $BAM $CHR":"$pos1"-"$pos2 | awk -f tools/findClips.awk \
     | sort -n | uniq -c | awk -f tools/countClips.awk "chr=$CHR" >> $RES;
  done
done

VCF=data_plat/NA12878.vcf.gz
RES2=data_plat/clips/NA12878_clusters.bed
> $RES2

#grep -w chr[0-9]* $FAI | while read -r CHR LEN x y z; do
grep -w chr1 $FAI | while read -r CHR LEN x y z; do
  bcftools view -H -r $CHR $VCF | awk -f tools/findClusters.awk >> $RES2
done

RPT=ref/repeats_dust.bed

#grep -w chr[0-9]* $FAI | while read -r CHR LEN x y z; do
grep -w chr1 $FAI | while read -r CHR LEN x y z; do
  RES=data_plat/clips/NA12878_clips_$CHR.bed
  sort -k2n $RES | uniq  > data_plat/tmp.bed
  grep -w $CHR $RES2 >> data_plat/tmp.bed
  bedtools sort -i data_plat/tmp.bed > data_plat/tmp2.bed
  bedtools intersect -a data_plat/tmp2.bed -b $RPT -v \
   | cut -f-3 > data_plat/beds/NA12878_regs_$CHR.bed
done
```

Reassemble the loci with SvABA and run the FPA algorithm on the contigs At each remaining candidate locus, the modified version of SvABA was used to extract the overlapping reads and assemble them into contigs. Resulting contigs were aligned to the reference genome and for contigs showing two or more base differences a TSM solution was computed. The output of the SvABA-FPA contain information to locate the locus in the genome, the alternative allele assembled by SvABA, and the FPA statistics explaining the properties of the candidate hit.

[ ]:
```
#REF=ref/GRCh38_full_analysis_set_plus_decoy_hla.fa.gz
REF=ref/hg38.no_alt.chr1.fa
bwa index $REF

FAI=$REF.fai
BAM=data_plat/NA12878.bam
OUTD=results_sr/NA12878_plat

mkdir $OUTD
#grep -w chr[0-9]* $FAI | while read -r CHR LEN x y z; do
grep -w chr1 $FAI | while read -r CHR LEN x y z; do
  BED=data_plat/beds/NA12878_regs_$CHR.bed
  OUT=$OUTD/NA12878_regs_$CHR
  ./tools/svaba run -v2 -t $BAM -G $REF -k $BED -a $OUT --fpa -r all &> $OUT.stdout
done
```

Filter the TSM hits The search finds large numbers of hits. To enrich the convincing cases and to remove e.g. overlaps with microsats and other low-complexity sequence (some of which probably are true TSMs), several steps of filtering were performed. The steps are implemented in the awk-script `tools/filterFpaHits.awk`. Some of the conditions are: solutions intersecting with masked sequence or assembled contigs having low sequence complexity (Trifonov's complexity with order 5 > 0.25; computed with program SeqComplex) were removed; TSMs were compared to forward alignments and required to be better, containing at least two edit events (of which at least one base mismatch) less than the non-template-switching alignment; of those, ones with (1)-(4) distance longer than 250 bp or shorter than 5 bp and upstream/downtream flanking regions or (2)-(3) fragment showing sequence identity below 95% or being less than 10 bp long were discarded.

The filtering script creates several files of which the last, ending "_7", contains the hits that have passed all the filtering steps. Those are taken for the next step of the analysis.

[ ]:
```
#REF=ref/GRCh38_full_analysis_set_plus_decoy_hla.fa.gz
REF=ref/hg38.no_alt.chr1.fa
OUTD=results_sr/NA12878_plat
RPT=ref/repeats_dust.bed.gz

#grep -w chr[0-9]* $FAI | while read -r CHR LEN x y z; do
grep -w chr1 $FAI | while read -r CHR LEN x y z; do
    INP=$OUTD/NA12878_regs_$CHR.fpa.out.gz
    awk -f tools/filterFpaHits.awk -v "inf=$INP" -v "bed=$RPT" -v "ref=$REF" \
                                   -v "chr=$OUTD/$CHR" -v "outf=$OUTD/NA12878.fpa"
done
```

Process the identified TSM loci and genotype them The details of the hits are given as comma-separated values, explained in some detail above. The first five fields of the SvABA-FPA output give the coordinates of the TSM event, and the third and fourth field (points (2) and (3), defining the end points of the copied fragment) allow selecting cases by the length. As a simple example, below TSM hits of at least 25 nuc are selected (the awk condition within the pipes), and then the alternative alleles are created using the awk-script `tools/createAllelesSwap.awk`. Given the alterntaive alleles, the loci are genotyped using short-read data and the python-scipt `tools/tsm_alleles2.py`. Given the alleles and locus information, it is trivial to genotype the TSM candidates with bams from different sequencing runs or from different individuals.

[ ]:
```
#REF=ref/GRCh38_full_analysis_set_plus_decoy_hla.fa.gz
REF=ref/hg38.no_alt.chr1.fa
HTS=results_sr/NA12878_plat/NA12878.fpa_7
FAS=results_sr/NA12878/alleles/NA12878_plat_25
RES=results_sr/NA12878/bams/NA12878_plat_25
BAM=data_plat/NA12878.bam
```

```
mkdir -p $FAS

cat $HTS | awk '$3-$4+1>24' | awk -f tools/createAllelesSwap.awk "ref=$REF" "fas=$FAS"

mkdir -p $RES

python3 tools/tsm_alleles2.py -a -s -d $FAS -r $BAM -b $RES -o -x \
 > $RES/results.txt
```

The same approach for genotype posterior probabities was used within R. In this example, we have identified the TSM candidate loci in Platinum data and resolved the genotypes in the same data only.

[ ]:
```
dat.plat <- read.table("results_sr/NA12878/bams/NA12878_plat_25/results.txt")
colnames(dat.plat)  <-  c("chrom","start","stop","dur","dir","ddr","dua","dia","dda")

keep <- filter(dat.plat,9)

gt.plat <- factor(score(dat.plat[keep,]),levels=c(0,1,2,3),labels=c("NA","RR","RA","AA"))

cbind.data.frame(dat.plat[keep,1:3],gt.plat)
```

Validation of TSM loci in PacBio sequencing data and 1000G variant data

Platinum and Chromium data contain TSM-like loci that are not reported in HaplotypeSV data. Two types of cases were found: (i) the variants are not reported in full and the reconstructed REF allele is erroneous, not allowing a TSM solution between the alleles; and (ii) variants at the locus are completely missing.

The loci were validated using PacBio long-read data and 1000G variant data.

[ ]:
```
BAM=https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/NA12878/PacBio_SequelII_CCS_11kb/
 ↪HG001_GRCh38/HG001_GRCh38.haplotag.RTG.trio.bam
REF=ref/hg38.no_alt.chr1.fa

for POS in chr1:68086409 chr1:88767997 chr1:177645239 chr1:223759874 ; do
 echo -e $POS"\n"
 samtools tview $BAM $REF -p $POS -d T | head -20
 echo
done
```

[ ]:
```
for POS in chr1:68086419-68086450 chr1:88768007-88768051 chr1:177645249-177645288 chr1:
 ↪223759884-223759915; do
 CHR=${POS/:[0-9|-]*/};
 echo -e "1000 Genomes "$POS"\n"
 VCF=http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000G_2504_high_coverage/working/
 ↪phase3_liftover_nygc_dir/phase3.$CHR.GRCh38.GT.crossmap.vcf.gz
 bcftools query -r $POS -sNA12878 -f '%CHROM\t%POS\t%REF\t%ALT\t%INFO/AF\t\t%INFO/EUR_AF\t[%GT]\n' $VCF |␣
 ↪column -t;
 echo
done
```

[ ]:
```
VCF=data_sv/freeze3.snv_indel.alt.chr1.vcf.gz
for POS in chr1:68086419-68086450 chr1:88768007-88768051 chr1:177645249-177645288 chr1:
 ↪223759884-223759915; do
 echo -e "HaplotypeSV "$POS"\n"
 bcftools query -r $POS -sNA12878 -f '%CHROM\t%POS\t%REF\t%ALT\t[%GT]\n' $VCF | column -t;
 echo
done
```

Analysis of gnomAD and Sasani et al. data

Identify and analyse TSMs in gnomAD data First, clusters of at least three variants are identified. Given these, the script `tools/tsm_scan_dnm.py` does the TSM scan and visualises the hits. Some processing is done to find hits that are longer than 25 bp (and to select the longest ones of the possibly many hits triggered by nearby MNVs) and then to reorder the output into the final format. The (2)->(3) regions are extracted in a similar manner and tandem repeats are searched with `trf`.

To save space and time, the data are provided and the analysis is performed only for chr1.

[ ]:
```
mkdir results_dnm

#for i in `seq 1 22` X Y; do
# python3 tools/tsm_scan_dnm.py -f ref/human_g1k_v37.fasta -g data_dnm/gnomad_mnv_genome_filt_clust3.tsv␣
 ↪-c $i \
#  > results_dnm/gnomad_tsm_chr$i.txt &
#done

python3 tools/tsm_scan_dnm.py -f ref/human_g1k_v37.chr1.fasta -g data_dnm/gnomad_mnv_genome_filt_clust3.
 ↪tsv -c 1 \
  > results_dnm/gnomad_tsm_chr1.txt &

for i in `seq 1 22`; do
 grep -A1 sample results_dnm/gnomad_tsm_chr$i.txt | tr -d '\n' | sed 's/sample/\nsample/g' \
  | awk -F, '$8-$9+1>24 && $8-$9+2==$10-$7{if($8-$9>l[$1]){l[$1]=$8-$9;h[$1]=$0}}END{for(i in h){print␣
 ↪h[i]}}' \
  | cut -d" " -f-5 |sort -V | while read j k l m n; do
  grep "$j $k $l $m $n" results_dnm/gnomad_tsm_chr$i.txt | head -1 | while read a b c d e f; do
   cat data_dnm/gnomad_mnv_genome_filt_clust3.tsv | awk '$1==c && $2>=d && $2<=e' c="$c" d="$d" e="$e" \
   | sort -V | uniq
  done
  grep -A19 "$j $k $l $m $n" results_dnm/gnomad_tsm_chr$i.txt | perl -pe 's/masked\n/masked /' | \
  awk -F, '{if($1~"sample"){if(l>longest){buffer=b; longest=l};l=$8-$9;b="";ins=0}
  if($1~"Switch"){ins=1}
  if(ins>0){b=b"\n"$0}}
  END{if(l>longest){buffer=b}print buffer}'
 done
done > results_dnm/gnomad_tsm_out.txt &

for i in `seq 1 22`; do
 grep -A1 sample results_dnm/gnomad_tsm_chr$i.txt | tr '\n' ' ' | sed 's/sample/\nsample/g' \
  | awk -F, '$8-$9+1>24 && $8-$9+2==$10-$7{if($8-$9>l[$1]){l[$1]=$8-$9;h[$1]=$0}}END{for(i in h){print␣
 ↪h[i]}}' \
  | cut -d" " -f-5 | sort -V | while read j k l m n; do
  grep -A19 "$j $k $l $m $n" results_dnm/gnomad_tsm_chr$i.txt | perl -pe 's/masked\n/masked /' | \
  awk -F, '{if($1~"sample"){if(l>longest){buffer=b; longest=l};l=$8-$9;b="";ps=$9;pe=$8;ins=1;
 ↪split($1,a," ")}
  else if(ins==1){b=">"a[3]"_"a[4]"_"a[5]"\n"substr($0,ps,pe-ps);ins=0}
  }
  END{if(l>longest){buffer=b}print buffer}'
 done
done > results_dnm/clust3_23.fa

trf results_dnm/clust3_23.fa 2 7 7 80 10 20 4 -h -ngs > results_dnm/clust3_23_trf.txt
```

Identify and analyse TSMs in Sasani et al. data The result may depend on the parameter "scan_flank" in `tools/tsm_scan_dnm.py`.

[ ]:
```
python3 tools/tsm_scan_dnm.py -f ref/human_g1k_v37.fasta -g data_dnm/third_gen.dnms.tsv > results_dnm/
 ↪third_gen.dnms.out
```