milesjor / VAULT Private

Variant Analysis with UMI for Long-read Technology (VAULT) Edit

Manage topics

2 commits | 1 branch | 0 packages | 0 releases | GPL-3.0

Branch: master ▾ | New pull request | Create new file | Upload files | Find file | Clone or download ▾

Chongwei add pic of 5' 3' umi		Latest commit 598bd45 7 minutes ago
example	add pic of 5' 3' umi	7 minutes ago
tools	Initial commit	13 minutes ago
.DS_Store	Initial commit	13 minutes ago
.gitattributes	Initial commit	13 minutes ago
LICENSE	Initial commit	13 minutes ago
README.md	add pic of 5' 3' umi	7 minutes ago
__init__.py	Initial commit	13 minutes ago
_version.py	Initial commit	13 minutes ago
check_umi.sh	Initial commit	13 minutes ago
variants_calling.py	Initial commit	13 minutes ago
vault.py	Initial commit	13 minutes ago

README.md

VAULT

Variant Analysis with UMI for Long-read Technology (VAULT)

VAULT is a tool for analyzing UMI-labeled reads, works for both error-prone long reads and accurate single-end/paired-end short reads.

More detail: Long-read Individual-molecule Sequencing Reveals CRISPR-induced Genetic Heterogeneity in Human ESCs

INSTALL

Prerequisites

Anaconda (https://www.anaconda.com/distribution/) or Miniconda (https://conda.io/miniconda.html). For example, users may download and install the following Anaconda3 package:

```
wget https://repo.anaconda.com/archive/Anaconda3-2019.10-Linux-x86_64.sh
bash Anaconda3-2019.10-Linux-x86_64.sh
```

Download the VAULT package

```
git clone https://github.com/milesjor/VAULT.git
cd ./VAULT/
```

Install all required modules

```
conda env create --name vault --file ./tools/vault_env.yaml
conda activate vault
python -m pip install numpy
python -m pip install pandas
```

Parameters

The parameters of VAULT

Users can use the following command to print out the parameters:

```
python ./vault.py -h
```

A list of available parameters:

```
optional arguments:
  -h, --help            show this help message and exit

Required options:
  -u UMI_ADAPTER, --umi_adapter UMI_ADAPTER                           # UMI primer sequence
                        UMI sequence, automatically detect \
                        NNNATGCNNN as UMI
  -s SAVE_PATH, --save_path SAVE_PATH
                        path/to/save/
  -r REFER, --refer REFER                                             # the reference for alignment, \
                                                                        should be the amplicon sequence
                        path/to/ref.fa
  -q FASTQ, --fastq FASTQ
                        path/to/reads.fastq, or fastq.gz

Optional options:
  -e ERROR, --error ERROR                                             # related to raw read error rate
                        error tolerance rate for umi analysis [0.11]
  -t THREAD, --thread THREAD                                          # thread for processing UMI
groups
                        thread/process number [5]
  -T THRESHOLD, --threshold THRESHOLD                                 # ignore UMI group with read
number <= [5]
                        Threshold of read number for snp analysis [5]
  -b BASH_THREAD, --bash_thread BASH_THREAD                           # thread in variant calling
                        Thread for running bash cmd [1]
  -p PE_FASTQ, --pe_fastq PE_FASTQ
                        read2.fastq for illumina pair-end sequencing
  -a {sr,map-ont,map-pb}, --align_mode {sr,map-ont,map-pb}            # [sr] for Illumina, [map-ont]
for Nanopore,\
                                                                        [map-pb] for PacBio
                        parameter in alignment, minimap2 -ax \
                        [sr|map-ont|map-pb]
  --unmapped_reads      extract mapped reads before UMI analysis      # will filter out reads that
cannot align to reference
  -v, --version         show the current version
```

Example

Command

```
python ./vault.py -u CATCTTACGATTACGCCAACCACTGCGGNNNNNTGNNNNNGACACATTCTCCCAGGCCCTACTT \
                 -q ./example/nanopore_reads.fastq.gz \
                 -s ./example/result \
                 -r ./example/reference.fa \
                 -e 0.11 \
                 -a map-ont \
                 --unmapped_reads \
                 -t 4 \
                 -b 1
```

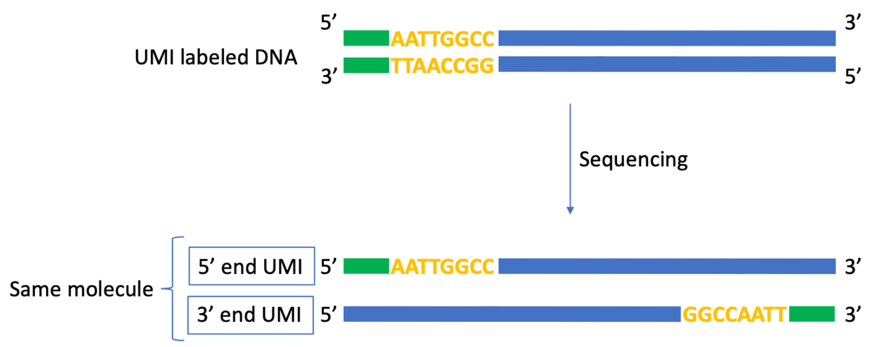
Results

```
./result/
├── nanopore_reads_alignment_summary.log    # raw reads alignment summary
├── nanopore_reads.bam
├── nanopore_reads.bam.bai
├── nanopore_reads.mapped.fastq             # reads that can map to reference
├── nanopore_reads.mapped.lst
├── nanopore_reads.sam
├── grouped_reads                           # fastq reads for every UMI groups
│   └── perfect_umi
├── snp
│   ├── all_snp_from_perfect_umi.vcf        # raw variant calling result for small variants
│   ├── all_sv_from_perfect_umi.vcf         # raw variant calling result for large variants (>=30bp)
│   ├── flt_sv_from_perfect_umi.vcf         # filtered variant calling result for large variants
(>=30bp)
│   ├── pass_snp_from_perfect_umi.vcf       # filtered variant calling result for small variants
│   ├── all_sv_from_perfect_umi.filtered.0.5.vcf           # customized SV filter result
│   ├── all_sv_from_perfect_umi.filtered.0.5.sorted.vcf    # customized SV filter result and sort by
position
│   └── perfect_umi
└── umi_analysis                            # UMI analysis result for 5' and 3' end of reads
    ├── 3end_UMIs
    └── 5end_UMIs
```

Individual UMI group folder

```
./result/snp/perfect_umi/
├── 14_ATCGATGATTTT_AAAATCATCGAT    # 14 reads in this group, 5' UMI is ATCGATGATTTT, 3' is
AAAATCATCGAT
├── 33_GACATTGTCTGG_CCAGACAATGTC
├── 35_5end_AACAGTGCTGCT            # 35 reads in this group, all reads from 5' UMI
├── 5_3end_AAAAACATGGCA             # 5 reads in this group, all reads from 3' UMI
├── 7_5end_ATTCTTGGTGTC
├── 7_CTATGTGAAGAA_TTCTTCACATAG
├── 8_3end_ACAAGCAAAAAA
├── 8_AGTTGTGCCATA_TATGGCACAACT
├── 8_CCGCGTGAGATG_CATCTCACGCGG
├── 8_CGTTGTGTTACT_AGTAACACAACG
├── 8_CTATTTGTCACT_AGTGACAAATAG
├── 8_GGGTTTGGTTTG_CAAACCAAACCC
├── 8_GTGGGTGACGGG_CCCGTCACCCAC
├── 8_GTGTTTGTTAGA_TCTAACAAACAC
├── 8_TCAATTGCAGAA_TTCTGCAATTGA
├── 8_TTACTTGATTTT_AAAATCAAGTAA
├── 8_TTGGATGGAAGT_ACTTCCATCCAA
├── 9_AAAGATGCGCGT_ACGCGCATCTTT
├── 9_AGAAATGATAGC_GCTATCATTTCT
├── 9_ATCGATGGTGCG_CGCACCATCGAT
├── 9_ATGTTTGCCAAT_ATTGGCAAACAT
├── 9_CTAACTGCTTAT_ATAAGCAGTTAG
├── 9_GAGAATGAGTAC_GTACTCATTCTC
└── 9_GTTTATGTACAT_ATGTACATAAAC
```

5' end UMI and 3' end UMI


UMI labeled DNA
5’
AATTGGCC
3’
3’
TTAACCGG
5’
Sequencing
Same molecule
5’ end UMI
5’
AATTGGCC
3’
3’ end UMI
5’
GGCCAATT
3’