Dindel User Guide, version 1.0

Kees Albers
University of Cambridge, Wellcome Trust Sanger Institute
caa@sanger.ac.uk

October 26, 2010

Contents

1 Introduction

Dindel is a program for calling small indels from next-generation sequence data by realigning reads to candidate haplotypes. The simplest use of Dindel is to consider all in indels in a read-alignment file (a BAM file), and to test whether each of these is a real indel or a sequencing error or mapping error. Dindel can also test candidate indels or sequence variants from other sources, e.g. a database of known variants, or indels called on a different sample for which you want to know if it is also supported by reads in your current sample.

Please read the section on Filtering and Important Dindel parameters before you use Dindel.

2 Requirements

To use Dindel, you need at least the following two:

1. A file with read-alignments: Dindel currently accepts only BAM (Li et al., 2009) files as input. The BAM file must be indexed using SAMtools (Li et al., 2009) to produce a valid `.bai` file. Most read mappers nowadays produce BAM files as output.

2. The reference sequence file used to align the reads in the BAM file. The reference sequence file must be in Fasta format, and must be indexed using Samtools to produce a valid `.fai` file.

Then Dindel can extract all indels from the read-alignment file, and subsequently realign reads around these candidate indels to see if they are supported by multiple reads. This basic workflow is outlined in section 4; examples are given in section 5.1 for diploid samples and 5.2 for pooled data, below.

Platform: Currently, Dindel only analyzes Ilumina GA data properly. Dindel uses an sequencing error indel model that has been trained on Illumina data. It is not appropriate for 454 data.

Mapping qualities: Dindel explicitly uses mapping qualities of reads in its probabilistic realignment model. It is therefore essential to use a read mapper that produces well calibrated mapping qualities for the read alignments. Stampy (`http://www.well.ox.ac.uk/project-stampy`), BWA (Li and Durbin, 2009) and MAQ (Li et al., 2008) are examples of read mappers that provide useful mapping qualities (we have not explicitly tested others).

Unmapped reads: Dindel can realign unmapped mates of mapped reads. For longer insertions and deletions this may signficantly improve the power of Dindel. For this it is essential that Dindel is provided with correct library insert sizes.

3 Optional input

In a slightly more advanced analysis, one can augment the candidate indels obtained from the BAM file with a set of candidate indels specificied by the user. This may include known indels (eg dbSNP indels, or 1000 Genomes indels) as well as SNPs.

One excellent way to augment the set of candidate indels is by running Pindel (Ye et al., 2009), or testing small indel variants obtained from de novo assembly, or indels from a different technology (eg 454).

4 Dindel workflow

The Dindel workflow can be divided into four stages:

Stage 1 In the first stage Dindel extracts all indels from the read-alignments in the BAM file. These indels are the *candidate* indels around which the reads will be realigned in stage 3.

In this stage, Dindel also infers the library insert size distributions. These will be used in stage 3 for paired-end reads.

Stage 2 The indels obtained in stage 1 from the BAM file are the *candidate* indels; they must be grouped into windows of ~ 120 basepairs, into a `realign-window-file`. The format of this file is described in section 15. The included Python script `makeWindows.py` will generate such a file from the file with candidate indels inferred in the first stage.

Stage 3 Then, for every window, Dindel will generate candidate haplotypes from the candidate indels, and SNPs it detects in the BAM file, and realign the reads to these candidate haplotypes. The realignment step is the computationally most intensive step.

Stage 4 The last stage consists of interpreting the output from Dindel and produces indel calls and qualities in the VCF4 format.

5 Examples

5.1 Example 1: The basic procedure for calling indels from a diploid sample

We assume we have a BAM file `sample.bam` and a reference file `ref.fa`, and the corresponding index files `sample.bam.bai` and `ref.fa.fai` as generated by SAMtools.

Stage 1 We extract indels from the BAM file and simultaneously infer the library insert size distribution using:

```
dindel --analysis getCIGARindels --bamFile sample.bam \
       --outputFile sample.dindel_output --ref ref.fa
```

This will produce a file `sample.dindel_output.variants.txt`, which contains all *candidate* indels, and the file `sample.dindel_output.libraries.txt`, which contains the insert size distribution for all libraries in the file.

Stage 2 The second stage consists of making the realignment windows. The file with the indels `sample.dindel_output.` cannot be used directly as input for the second stage. First we need to convert it to a `realign-windows-file`, with the following command:

```
makeWindows.py --inputVarFile sample.dindel_output.variants.txt \
               --windowFilePrefix sample.realign_windows --numWindowsPerFile 1000
```

This creates window-files `sample.realign_windows.1.txt`, `sample.realign_windows.2.txt`, ..., that can be used as input for the realignment option of Dindel. It is a good idea to put the realign-windows files in their own subdirectory/folder, as there can be many.

Stage 3 We now have all the input files needed to realign the reads. To realign all windows in `sample.realign_windows.` use the command:

```
dindel --analysis indels --doDiploid --bamFile sample.bam --ref ref.fa \
       --varFile sample.realign_windows.2.txt \
       --libFile sample.dindel_output.libraries.txt \
       --outputFile sample.dindel_stage2_output_windows.2
```

The `--doDiploid` ensures that Dindel analyses the BAM file as if all reads were sampled from two haplotypes. A separate analysis must be run for each realign-windows file created by the `makeWindows.py` script. Note that the `--outputFile` option specifies the *prefix* of the output files, not the full name of the output file.

Stage 4 The final step is to integrate the results from all windows into a single file with variant calls. The command for this is:

```
mergeOutputDiploid.py --inputFiles sample.dindel_stage2_outputfiles.txt \
                      --outputFile variantCalls.VCF --ref ref.fa
```

Here the file `sample.dindel_stage2_outputfiles.txt` should contain all `*.glf.txt` output files generated by Dindel at the second stage (in this example all files starting with the prefix

sample.dindel_stage2_output). The script will then generate a single VCF file from these files. The reference sequence is required to annotate variants against the reference sequence and filter out variants in long homopolymers (> 10).

When creating the VCF file several filters are applied. To change these, from the command line do `mergeOutputDiploid.py --help` for a list of options.

5.1.1 Ignoring read-pair information

The reads can be realigned ignoring library insert sizes. To do this simply omit the `--libFile` option in stage 2.

5.2 Example 2: The basic procedure for calling indels from a pool

The procedure for pooled analysis is similar to that of a single sample. It is possible to provide multiple BAM files for simultaneous analysis, however, Dindel will assume in that case that each BAM file corresponds to a separate individual when outputting genotype likelihoods. It does *not* affect the way haplotype frequencies, variant frequencies and the posterior probabilities are estimated, because for that calculation Dindel ignores which BAM file a read comes from.

Stage 1 We extract indels from the BAM file and simultaneously infer the library insert size distribution using:

```
dindel --analysis getCIGARindels --bamFile sample.bam \
       --outputFile sample.dindel_output --ref ref.fa
```

This will produce a file `sample.dindel_output.variants.txt`, which contains all *candidate* indels, and the file `sample.dindel_output.libraries.txt`, which contains the insert size distribution for all libraries in the file. If one has multiple BAM files, this command needs to be run for every BAM file separately.

Note This will produce one library insert size distribution for every BAM file. The user needs to merge these into a single file that can be used with the `--libFile` option.

Stage 2 Again, the file with the indels `sample.dindel_output.variants.txt` cannot be used directly as input for the second stage. First we need to convert it to a `realign-windows-file`, with the following command:

```
makeWindows.py --inputVarFile sample.dindel_output.variants.txt \
               --windowFilePrefix sample.realign_windows --numWindowsPerFile 1000
```

This creates window-files `sample.realign_windows.1.txt`, `sample.realign_windows.2.txt`, ..., that can be used as input for the realignment option of Dindel. It is a good idea to put the realign-windows files in their own subdirectory/folder, as there can be many.

If multiple BAM files are used, then this command should be run on the merged `.dindel_output.variants.txt` file
(for example, by doing `cat *.variants.txt > allvariants.output.txt'`, and then run the `makeWindows.py` script on `allvariants.output.txt`.)

Stage 3 We now have all the input files needed to realign the reads. At this stage, the main difference with the diploid analysis of a single sample is that the `--doDiploid` option should be replaced with the `--doPooled` option.

To realign all windows in `sample.realign_windows.2.txt`, use the command:

```
dindel --analysis indels --doPooled --bamFile sample.bam --ref ref.fa \
       --varFile sample.realign_windows.2.txt \
       --libFile sample.dindel_output.libraries.txt \
       --outputFile sample.dindel_stage2_output_windows.2
```

A separate analysis must be run for each realign-windows file created by the `makeWindows.py` script. Note that the `--outputFile` option specifies the *prefix* of the output files, not the full name of the output file.

Stage 4 The final step is to integrate the results from all windows into a single file with variant calls. In this case, the `--type` option is `pool`: The command for this is:

```
mergeOutputPooled.py --inputFiles sample.dindel_stage2_outputfiles.txt \
               --outputFile variantCalls.VCF --ref ref.fa
               --numSamples 50
               --numBamFiles 1
```

Here the file `sample.dindel_stage2_outputfiles.txt` should contain all `*.glf.txt` output files generated by Dindel at the second stage (in this example all files starting with the prefix `sample.dindel_stage2_output`). The script will then generate a single VCF file from these files. The reference sequence is required to annotate variants against the reference sequence and filter out variants in long homopolymers (> 10).

5.3 Example 3: Adding candidate variants

A crucial step that must be performed if candidate indels are added that were not extracted by Dindel from the BAM file in the first stage, is that the candidate indels must be repositioned to the left most position in the reference. Dindel has an option to do this:

```
dindel --analysis realignCandidates \
       --varFile <file with candidate variants> --outputFile <outputFile> --ref ref.fa
```

Note: The helper script `convertVCFToDindel.py` is able to convert indels in the VCF4 format to the file format required by the script `makeWindows.py`.

Note: Dindel performs this step automatically when it extracts candidate indels from the BAM file. However, it assumes that all indels provided in the input file at the second stage (specified with `--varFile`) have been repositioned correctly. Therefore, if you add your own candidate indels to these input file, you must make sure they are repositioned using this command, otherwise Dindel will not report them in the final VCF file. They will be reported in intermediate `.glf.txt` files, however, only indels with `was_candidate_in_window=1` will be reported in the final VCF file.

If you have manually added candidate indels to the file with variants produced by Dindel in the first stage, but they do not show up in the VCF, then you might need to reposition the candidate indel.

6 Reducing the number of candidate indels

For multiple high-depth samples in pools, the number of candidate variants may become very high. The script `selectCandidates.py` can help reduce the number of candidates by requiring that each candidate indel was seen at least twice. This script requires that the candidates are sorted on chromosome first,

then (numerically!) on position. Usage of the script is very simple:

```
selectCandidates.py -i sample.dindel_output.variants.txt \
                    -o sample.dindel_output.variants.mincount2.txt
```

By default, `selectCandidates.py` outputs all candidate indels seen at least twice. However, this can be changed using the option `--minCount`.

If the input file was generated directly by Dindel, the output file can be used directly with the `makeWindows.py` script. However, if you are combining variants from multiple BAM files, the different `.variants.txt` files should be concatenated and then sorted. Don't use the unique option in sorting, as this will remove the essential information of how often each indel was detected by the read mapper. If the input file contains variants from some other source, one should run a
`dindel --analysis realignCandidates` step first as described above.

Note that Dindel realigns the candidate indels first, so that inconsistent gap assignments by the read mapper are reduced somewhat.

7 Filters

In the INFO field of the VCF field, useful information can be found to further filter the calls and decrease the number of false positives (at the expense of losing true calls).

In whole genome sequencing experiments, many artifacts can be avoid by avoiding the indel to be seen on both strands. There are four INFO tags in the VCF file that record this:

NF,NR These are the most stringent. NF is the number of times the equivalent indel region was seen on the forward strand, allowing at most one mismatch between the read sequence and the alternative haplotype, and requiring at least 4 bp of additional reference sequence around the indel to be covered by the read. For a homopolyer, the equivalent indel region consists of the whole homopolymer. In general, it is the whole region in the reference where the indel might be positioned without changing the sequence of the alternative haplotype. NR is the equivalent for the reverse strand.

NFS, NRS These are less stringent, and require only that the position at which the indel was called, plus the indel, is covered by a read. The difference with NF is that NFS and NRS don't take into the repetiveness of the reference sequence at the position of an indel. So in the case of a homopolymer, NFS and NRS do *not* correspond to the number of reads spanning the homopolymer run, whereas NF and NR do.

Targeted resequencing: With targeted resequencing experiments, such as exome capture protocols, it is not possible to have reads on both the forward and the reverse strand at the boundary of a targeted region. Therefore these filters should be applied with care in those situations.

8 Output

Dindel produces several output files. In the second stage, Dindel produces `.glf.txt` files. These files are space delimited intermediate output files. They contain for every window the genotype likelihoods

for the candidate variants. Sometimes Dindel decides to skip a window, for instance when the number of reads is too high, or the number of candidate haplotypes becomes is too high. In the first column of each `.glf.txt` file Dindel will report if there was no error (`ok`) or the reason for skipping the window.
Note: All chromosomal coordinates reported in the `.glf.txt` files are zero-based.

The `mergeOuputDiploid.py` and `mergeOuputPooled.py` scripts merge all Dindel `.glf.txt` files, and creates a VCF file with the called variants. The coordinates in the VCF file are *one*-based. The examples show how to use this script to parse the output of Dindel files into a single file with variant calls.

Note: Dindel will report only candidate indels for which there is some evidence that they are real (quality score > 1).

For analysis of pools, the output files of Dindel do contain posterior probabilities. However, it is still recommended to use the `mergeOutputPooled.py` scripts to produce a single file with calls.

9 Analyzing multiple BAM files

It is possible to analyze multiple BAM files simultaneously with Dindel. Basically Dindel loads all BAM files and takes reads from each BAM file. Dindel can be fed multiple bamfiles by using the `--bamFiles` option: `dindel --bamFiles bamfiles.txt ...`
The file `bamfiles.txt` (any filename is allowed) should contain the full path of the different BAM files that should be jointly analyzed.

9.1 Diploid analysis

In the diploid analysis, Dindel will assume that all BAM files belong to the *same* individual.

9.2 Pooled analysis

In the pooled analysis, Dindel behaves differently. For calling the indels, Dindel will pool all reads and even if it is known which BAM file or read corresponds to which individual, Dindel will *not* use this information for calling the indels or estimating the allele frequency. If multiple BAM files are used, the `--numBAMFiles` option for `mergeOutputPooled.py` script should be specified.

However, if multiple BAM files are used, Dindel will output a genotype likelihood for every individual and indel in the intermediate `*.glf.txt` files. Only then Dindel will assume that each BAM file corresponds to a single diploid individual, in order to compute te genotype likelihoods. The genotype likelihoods for the called indels can be extracted using the `makeGenotypeLikelihoodFilePooled.py` script. This script requires the VCF file produced by `mergeOutputPooled.py`.
Type `makeGenotypeLikelihoodFilePooled.py --help` to see the required input parameters.

10 Output of realigned BAM files:

Currently Dindel does not support outputting a single realigned BAM file. However, it does have an option of outputting a realigned BAM file for every window that has been realigned, ie for every window specified in the realign-window-file provided to Dindel with the `--varFile` option. These BAM files are unsorted.

Callback on realigned BAM for SNP calling: Dindel creates a BAM file for every window. For a genome-wide sequenced sample, this could results in over $100,000$ files. Dindel provides a callback option: by specifying a script with the
`--processRealignedBAM` (to be precise, an absolute path to script), Dindel will call that script with five arguments: the first is the name of the realigned BAM file, the second is the Dindel `--outputFile` argument appended with `realigned`, so that the output can be *appended* to a file of choice. The final three arguments are the chromosome, leftmost coordinate of the window with sufficient coverage, and the rightmost with sufficient coverage. This script can be used to run for instance SAMtools pileup on the realigned BAM, although care should be taken that coverage near the edges of the window will be artificially lower due to boundary effects. The script may delete the realigned BAM file, to save space.

An example callback script (bash) that I have used is:

```
#!/bin/bash
samtools view $1 >> $2.sam
echo $3 $4 $5 >> $2.windows
rm $1
```

In this example the bamfile for the realignment window (stored in `$1`) is first appended to a SAM file starting with the prefix stored in `$2`, which is identical to the value specified for the Dindel `--outputFile` argument. Note that one must make sure that this file must not exist before Dindel is run! Then the coordinates of the window are appended to file that will contain all the coordinates of the windows. In the last line the temporary file `$1` is deleted again, so that the filesystem does not become cluttered with millions of files. When Dindel has been run, all SAM files are converted to BAM files, merged, sorted and indexed to produce a single BAM file.

`Warning` One needs to verify that the windows are not overlapping. If not, reads may be duplicated in the BAM file.

Warning: The `--outputRealignedBAM` option has not been extensively debugged. Currently output of realigned BAMs is *not* supported with the `--faster` option. In theory one could merge all realigned BAM files using SAMtools. However, it is possible that the same read may be included in two different realigned BAM files, so care should be taken when merging.

11 Important Dindel parameters:

A large number of Dindel parameters can be specified from the command line. Here we briefly describe the most important ones and how they potentially affect the results.

–bamFile,–bamFiles It is possible to provide more than one BAM file as input to Dindel. With the option `--bamFiles` (note the 's') it is possible to provide a list of BAM files to Dindel that should be

analysed jointly. How these BAM files are analyzed depends on the type of analysis, as described below.

Note: Although in principle Dindel can handle multiple BAM files, it is often better to merge BAM files into one single big BAM file, due to the memory requirements for loading the BAM index. This will be changed in future versions of Dindel.

–doDiploid Assume all input BAM files are from a single diploid individual. The Bayesian model explicitly assumes two haplotypes.

–doPooled Assumes the number of haplotypes is unlimited. Dindel estimates haplotype frequencies using a Bayesian EM algorithm.

Note that in this case, Dindel assumes that if the `--bamFiles` option is used, each BAM file corresponds to a seperate individual. In the output file, Dindel will report genotype likelihoods for each BAM file/individual, assuming that the data from each BAM file corresponds to a diploid individual. This step is performed *after* the Bayesian EM haplotype frequency estimation, and is equivalent to the genotype likelihood estimation of the `--doDiploid` option.

–maxRead This option limits the number of reads that is allowed per realignment window. This value should be adapted to something that is reasonable for your data set. Dindel will skip a realignment when this value is exceeded (and will say so in the output files).

–filterHaplotypes This options applies filtering to the set of candidate haplotypes, after realignment of reads to each candidate haplotype. Essentially, all haplotypes that have an indel that is not covered by at least x high-mapping quality reads 5 bp to the left and the right of region that defines the indel, are filtered and not considered in the Bayesian EM algorithm or diploid phasing algorithm. This option reduces sensitivity, but in some cases avoids artefacts. One example is when the two reads in a mate-pair must be aligned with a very small insert size.

–maxHap This option controls the number of candidate haplotypes against which each candidate variant is tested.

With `--maxHap 1`, each candidate indel is tested only against the reference haplotype. In this case, Dindel would create two candidate haplotypes: one being the reference haplotype, the other one being the reference haplotype with the indel added to it. Therefore, the total number of candidate haplotypes is equal to the number of candidate variants times the number specified after `--maxHap`.

The default and strongly recommended value for `--maxHap` is 8. This gives higher sensitivity and lower false-discovery rates on simulated data for longer indels. However, reasonable results may be obtained with `--maxHap 1`, but it will produce false indel calls near SNPs.

12 Strategies

12.1 Multiple samples

When dealing with multiple samples, there are different ways to analyse the data. Given the computational expense of Dindel, one approach is to first call for each sample indels independently. Then, one may combine a joint list of all indels called in all samples, and then test each indel again in each sample. This may reveal indels in sample that were not detected by the read mapper although the reads to support the indel.

12.2 Calling SNPs around indels

Although one could call SNPs on each realigned BAM file that Dindel produces for all windows, it can be more efficient to only perform a realignment around called indels, where in this case the threshold for calling an indel might be taken a bit lower than one would do normally.

13 Helper scripts

Dindel makes use of several Python scripts to preprocess and manage the input and output data.

makeWindows.py The `makeWindows.py` script takes variants as input and creates a file with groups of candidate variants (the 'windows'), in which reads will be realigned to candidate haplotypes consisting of those candidate variants. It distributes the windows over multiple files, so that the analysis can be run in parallel on a cluster.

mergeOutputDiploid.py The `mergeOutputDiploid.py` script merges the intermediate output files produced by after the Dindel stage for *diploid* analysis and creates a single annotated VCF4 file with indel calls.

mergeOutputPooled.py The `mergeOutputPooled.py` script merges the intermediate output files produced by after the Dindel stage for *pooled* analysis and creates a single annotated VCF4 file with indel calls.

convertVCFToDindel.py This script converts indels in VCF4 files to the variant format required by the `makeWindows.py` script. The variants obtained using `convertVCFToDindel.py` may be added to the candidates identified by Dindel using the `--getCIGARindels` option.

makeGenotypeLikelihoodFilePooled.py This script extracts the genotype likelihoods for the indels passing filters in the VCF file produced by `mergeOutputPooled.py`. These likelihoods can be further used in imputation software.

selectCandidates.py `selectCandidates.py` can reduce the number of candidate indels. See above for how this works.

14 Obtaining genotype likelihoods

The `*.glf.txt` files produced by Dindel contain genotype likelihoods which may be of interest to the users. To obtain these, one should take only lines with the proper value in the `analysis_type` column.

14.1 Single samples –doDiploid option

Lines where the third column equals `dip` contain proper genotype likelihoods. It is recommended to only use indels where the `was_candidate_in_window` column is '1' (the ninth column). Indels/candidate variants for which this column is zero are **not** reliable.

Lines where the third columns equals `dip.map` do not contain proper genotype likelihoods, but only the genotype *qualities* of the variants in the most likely (highest posterior probability) pair of haplotypes. These are conditional on there being an indel. These should *not* be used in imputation software.

14.2 Pooled samples –doPooled option

There is a helper script `makeGenotypeLikelihoodFilePooled.py` that extracts genotype likelihoods for the indels in the VCF file produced by `mergeOutputPooled.py`.

15 Format of input files

In this section we describe the requirements for the main input files for Dindel.

15.1 BAM file

The sequence aligment file should be coordinate sorted (using for instance the `samtools sort` command) and indexed (`samtools index <BAMfile>`).

15.2 Reference sequence

The reference sequence should be in plain-text Fasta format and should be indexed using SAMtools (`samtools index <Fasta reference file>`).

15.3 realign-window-file

The `Realign-window-file` is interpreted in a line-by-line fashion by Dindel. Each line in this file represents a region in which Dindel will realign reads to candidate haplotypes. The candidate haplotypes are generated by the candidate indels/variants specified in this file, and by additional variants that Dindel may infer directly from the BAM file, depending on the `--maxHap` parameter of Dindel (see section 16).

Each line must be formatted as follows:

```
 <chromosome> <left coordinate of window> <right coordinate of window> \
<variant 1> <variant 2> <variant 3> ..
```

A candidate variant should be specified as follows:
`<position>,<variant string>,[isCombinatorial],[allele frequency]`
Here, `position` should be the **zero-based** reference position of the variant. The convention for the position and the `variant string` is:

- `-XXXX` for a deletion. `X` can be anything. Dindel does not check whether the deleted sequence is actually consistent with the reference sequence.

The coordinate convention Dindel uses is:

`01234567` reference base-coordinates
`AATCGATG` REFERENCE
`AATC ATG` ALT: position=4, variant string=`-R` (or `-G`)

Thus, the position of the deletion is the coordinate of the *first* base in the *reference* sequence that is deleted from the reference.

- `+ATCC` for an insertion.

  `01234 567` reference base-coordinates
  `AATCG ATG` REFERENCE
  `AATCGTATG` ALT: position=5, variant string=`+T`

  Thus, the position of the insertion is the coordinate of the base in the reference *before* which the sequence is inserted.

- `R=>[ACGT]` for a SNP.

`isCombinatorial` is an indicator variable which should be either 0 or 1. 1 indicates that the candidate variant should be included combinatorially in the generation of candidate haplotypes.
`allele frequency`: this can be specified to specify a prior allele frequency for the candidate variant. It overrides the default prior probabilities that can be specified from the command line (see section 16). If `allele frequency` is specified `isCombinatorial` should also be specified.

A typical line in the `realign-window-file` looks as follows:
`1 7759867 7760023 7759927,-C 7759964,+G 7759941,-A 7759935,+A`
Here the chromosome is `1`, the left coordinate of the realignment window is `7759867`, the right coordindate is `7760023`, and there are four candidate indels. The coordinates of the candidate indels do not need to be sorted.

Note: The `realign-window-file` must be sorted first on chromosome (although the order of the chromosomes is not relevant), then sorted on the *left coordinate* of the realignment window. If the file is not correctly sorted, Dindel will abort.

16 Dindel parameters

Required	
–ref arg	fasta reference sequence (should be indexed with .fai file)
–outputFile arg	file-prefix for output results
–analysis arg (=indels)	Analysis type: **getCIGARindels**: Extract indels and library insert distribution from CIGARs of mapped reads **indels**: infer indels **realignCandidates**: Realign candidates in candidate file
–bamFile arg	read alignment file (should be indexed)
–bamFiles arg	file containing filepaths for BAMs to be jointly analysed (not possible for –analysis==indels
Required for analysis == indels:	
–varFile arg	file with candidate variants to be tested.
–varFileIsOneBased	coordinates in varFile are one-based
Options for analysis == getCIGARindels:	
	Region to be considered for extraction of candidate indels.:
–region arg	region to be analysed in format start-end, eg. 1000-2000
–tid arg	target sequence (eg 'X')
Output options:	
–outputRealignedBAM	output BAM file with realigned reads
–processRealignedBAM	absolute path to script to process realigned BAM file for each window
–quiet	quiet output
Parameters for analysis==indels option:	
–doDiploid	analyze data assuming a diploid sequence
–doPooled	estimate haplotype frequencies using Bayesian EM algorithm. May be applied to single individual and pools.
General algorithm parameters:	
–faster	use faster but less accurate (ungapped) read-haplotype alignment model
–filterHaplotypes	prefilter haplotypes based on coverage
–priorSNP arg (=0.001)	prior probability of a SNP site
–priorIndel arg (=0.0001)	prior probability of an indel site

–maxHap arg (=8)	maximum number of haplotypes in likelihood computation
–maxRead arg (=1000)	maximum number of reads in likelihood computation
–mapQualThreshold arg (=0.01)	lower limit for read mapping quality
–capMapQualThreshold arg (=100)	upper limit for read mapping quality in observationmodelold (phred units)
–capMapQualFast arg (=40)	cap mapping quality in alignment using fast ungapped method (WARNING: setting it too high (¿50) might result in significant overcalling!)
–skipMaxHap arg (=200)	skip computation if number of haplotypes exceeds this number
–maxReadLength arg (=500)	maximum length of reads
–maxHapReadProd arg (=100000)	skip if product of number of reads and haplotypes exceeds this value
Parameters for –doPooled option:	
–bayesa0 arg (=0.001)	Dirichlet a0 parameter haplotype frequency prior
–bayesType arg (=singlevariant)	Bayesian EM program type (all or singlevariant or priorpersite)
General algorithm filtering options:	
–checkAllCIGARs arg (=1)	include all indels at the position of the call site
Observation model parameters:	
–pMut arg (=1e-05)	probability of a mutation in the read
–maxLengthIndel arg (=5)	maximum length of a *sequencing error indel* in read
Library options:	
–libFile arg	file with library insert histograms as generated by –analysis getCIGARindels
Misc results analysis options:	
–compareReadHap	compare likelihood differences in reads against haplotypes
–compareReadHapThreshold arg (=0.5)	difference threshold for viewing
–showEmpirical	show empirical distribution over nucleotides
–showCandHap	show candidate haplotypes for fast method
–showHapAlignments	show for each haplotype which reads map to it
–showReads	show reads
–inferenceMethod arg (=empirical)	inference method
–opl	output likelihoods for every read and haplotype

References

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

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup 2009. The Sequence Alignment/Map (SAM) Format and SAMtools. *Bioinformatics*, page btp352.

Li, H., Ruan, J., and Durbin, R. 2008. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11):1851–1858.

Ye, K., Schulz, M. H., Long, Q., Apweiler, R., and Ning, Z. 2009. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, page btp394.