Using rbambools package

Wolfgang Kaisers, CBiBs HHU Dusseldorf

June 20, 2014

A short notice in advance: During the last release change, the variable 'nGapAligns' was replaced by 'nAlignGaps' because the new term describes the contained value less ambiguously than the old one.

1 What this package is made for

BAM files are a important and powerful file format in Bioinformatics. This package pursues several objectives:

- Provide a technical (reading and writing) access to BAM files from within R.
- Give an authentic representation of the informational structure inside BAM files as programming interface.
- Provide a fast, C-based access to special (cumulative) aspects of the stored information.

These objectives transform into three implementational layers:

- The samtools C-library (written by Heng Li).
- C-based align and align-gap container.
- A R S4 class library.

The samtools library is (almost) a copy of the library originally written by Heng Li. All reading and writing transactions are done via samtools. There is C-code which handle align data for whole ranges and C-code for accumulation of information about splice-sites from gapped aligns.
The R-part of the code contains objects which communicate directly with samtools for reading and writing files, managing of file-header data, managing data for single aligns and functions which transform align data into data.frame format. Then there are objects that calculate and keep align-gap information for whole BAM-files and to summarize align-gap data over several BAM-files.
Align-gaps are emphasized here because they are highly informative representations of genomic splice-sites in RNA-seq data.

2 SAM file format

Data in BAM files is compressed and optionally indexed data in SAM file format. The current definition of the SAM file format [2] can be found here:

`http://samtools.sourceforge.net/SAM1.pdf`.

BAM files contain sequence alignment data which is the result of potentially incomplete matching sequence snippets to a reference sequence. In practice the snippets are DNA sequences which come from short read sequencing of DNA or RNA extracted from a biological probe and the reference sequence is a genome reference. Usually one BAM file contains align data from one biological probe where the read number is in the magnitude of 100 million reads. The size of the corresponding compressed files is in the range of 10 Gbyte. A very important feature of BAM files is that sorted BAM files can be indexed and indexed files allow random access. This allows very fast access to aligns that are located in arbitrary regions of the reference genome.

BAM files are divided in a header section and an alignment section.

2.1 The header section

The header section contains the following information:

Tag	Description	Information
HD	Header line	Format version and sorting
SQ	Reference sequence dictionary	Indexed reference sequences (Chromosomes)
RG	Read group	Sequencing technology
PG	Program	Alignment program
CO	Comment	

There are accessor functions in this package for reading and writing the listed fields. The header section is stored and retrieved as binary structure (`bamHeader`) which is converted into a tag delimited string representation (`bamHeaderText`). All processing steps on BAM-header data work on the string representation. `rbamtools`-objects parse and compose strings from and to object slots which then can be accessed via script code.

2.1.1 The reference sequence dictionary

The reference sequence dictionary section contains a list of reference sequences (usually chromosomes). Off the six fields (declared in the SAM file format specification) usually only two are used:

Tag	Description
SN	Reference sequence name
LN	Reference sequence length

The reference sequence dictionary section misses an index entry (refid) which is used in alignment structures and is described below (2.2.1).

2.2 The alignment section

The alignment section contains a series of align datasets. Each align describes the coordinates of the identified sequence matches in the reference sequence. The information for each align basically consists of:

Field	Content
QNAME	Align name (read identifier)
RNAME	Reference sequence identifier
POS	Mapping position: 0-based
CIGAR	Matching type string
FLAG	A set of bitwise flags.

2.2.1 The RNAME identifier: refid

Although RNAME associates with a textual entry, usually this field contains a number which identifies a sequence in the header section. To make things complicated, RNAME is a "0-based" sequential identifier which is not explicitly included in the "Reference sequence dictionary" (SQ). So, RNAME=0 means the first SQ entry and the "0" is not present in the header. We call this missing value *refid* throughout this document and there are functions in this package that automatically generate and use this id. The refid value is used by the `samtools` library as sequence identifier in align-structures and for defining ranges in index based random access.

2.2.2 Position

The position entry gives the align start position. In order to check the analogy between query and reference sequence see the given position in refid defined string.
In order to find the exact matching position it's necessary to notice the base of the position notation. We distinguish "0-based" and "1-based" position notations. They differ by the index of the starting position (and therefore all positions).The first position in a "0-based" notation is 0 whereas the first position in a "1-based" notation is 1:

0-based	0	1	2
1-based	1	2	3

Both notations appear in samtools which makes the correct handling somehow confusing. The SAM file format specification says ([2], section 1.4): "POS: 1-based leftmost mapping POSition of the first machting base". Samtools source code comments (bam.h, line 164) state the contrary: "pos 0-based leftmost coordinate". As to expreriences with "tophat 2.0.0" and annotation data (Ensembl and UCSC), the latter seems to be true.

In order to reflect the technical file content, two functions (`position` on `bamAlign` objects and `as.data.frame` on `bamRange` objects) return the file contained value (which is 0-based). In order to get values that are congruent with annotation (and IGV genome-browser data) the position values have to be increased by one.

The `bamGapList` objects which operate on align gaps contain "1-based" positions. So, overlapping with annotation data, can be done without correction.

2.2.3 Navigation on reference sequence

Printing the reference sequence results in characters that are ordered from left to right in ascending order of their position coordinate (consistent with ordinary reading succession). We refer to this image when two or more locations are compared. Lower coordinates are assumed to be on the "left" side and higher coordinates are assumed to be on the "right" side.

So, for genes coding on the "+" strand, "left" would be synonymous to "upstream" and "right" would be synonymous to "downstream".

2.2.4 CIGAR string

Alignments algorithms usually tolerate to some extend inexact matching. The type of matching is described in the CIGAR string (see [2] 1.4, Nr. 6). The CIGAR string is made up of CIGAR-items. A CIGAR-item consists of a integer number and a character. The number counts the affected positions (cigar-length). The character describes the type of operation (cigar-type). The following table shows relevant operations:

Operation	Label	Description
M	Match	Exact match of x positions
N	Alignment gap	Next x positions on ref don't match
D	Deletion	Next x positions on ref don't match
I	Insertion	Next x positions on query don't match
		(x = cigar-length)

The operations "N" and "D" are mechanistic identical but they describe biological different entities: "D" means genomic deletions, where few nucleotides on the genome get lost whereas "N" means gaps which occur in RNA-seq alignments. These gaps are due to DNA-splicing events and their size can achieve magnitude of $10^3 - 10^5$.

First example: The shown alignment is an exact match and will give `position = 2` (0-based!) and `CIGAR = 6M`:

```
AAGTCTAGAA (ref)
  GTCTAG   (query)
```

Second example: We see an alignment with two nucleotides ("GA") inserted into the reference. The align entries will be `position=3` (0-based!) and `CIGAR=3M2I2M`:

```
AAAGTCGATGAA (ref)
   GTC  TG   (query)
```

Third example: Here we have a deletion on the reference. The ”C” in the query sequence has no match. The align entries will be `position=3` and `CIGAR=2M1D3M`:

```
AAGT TAGAA (ref)
  GTCTAG   (query)
```

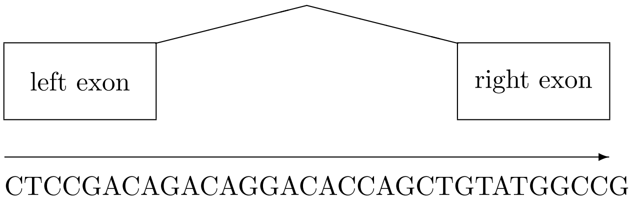
Fourth example: This is a gapped alignment (due to a splicing event in RNA-seq). It will give the entries `position=3` and `CIGAR=3M7N4M`:

```
CCCTACGTCCCAGTCAC (ref)
   TAC       TCAC (query)
```

We see the alignment gap (”GTCCCAG”). From the ”GT” and ”AG” at the gap boundaries, one can assume that this splice-site is on the ”+” strand.

2.3 Gapped alignments

A special focus of functionality inside this package are Alignment gaps. Alignment gaps in RNA-seq experiments are viewed as phenomenons that rely on biological splicing mechanisms during protein-biosynthesis and the resulting exon-intron structure of the genome.



2.4 Gap-sites

Gap-sites are alignment gaps (=gap-regions) that are shared by one or more aligns. The nucleotides on the reference sequence that are skipped in the alignment (i.e. the reference region which is depicted by ”N” cigar items) form the gap-region. Gap-sites are also characterized by the fact that they are bordered by M-segments on either side. The amount of information about the existance of gap-site in the alignment is proportional to the number of matching nucleotides that make up the framing M-segments. The calculated derived values on gap-sites therefore center on three measures:

- The number of aligns that define the gap-site.

- The Length of the framing M-segments.
- The number of different length values in the framing M-segments.
- The number of alignments (probes, number of BAM-files) in which the gap-site is found.

Gap-sites are of special interest in RNA-seq experiments because they arise from mRNA sequence which spans a processed splice site (splicing results in removal of intronic sequence ranges from pre-mRNA). Gapped alignments contain highly specific information about splicing events. Of central interest in RNA-seq experiments is the identification and quantification of splicing events.

In order to describe and illustrate the parameters that are calculated and kept within this package we show the following:

2.4.1 Example

The following table example shows a short reference sequence and three different alignments that define a gap-site. The reference nucleotides that consitute the gap-region are printed in red:

	qname	position	CIGAR
`   AG      CCTTGATG`	align1	3	2M6N8M
`  CAG      CCTTGAT`	align2	2	3M6N7M
` CCAG      CCT`	align3	1	4M6N3M
`CCCAGGTCCAGCCTTGATGTCC`	(reference)	(0-based)	

For each gapped align from which the gap-site is constituted, three values concerning the number of matching nucleotides are kept:

- **lcl** (left cigar length) is the length of the left adjacent match in the CIGAR string.
- **rcl** (right cigar length) is the length of the right adjacent match in the CIGAR string.
- **mcl** (minimum cigar length) is the minimum of the `lcl` and `rcl` value for each align.

For these parameters we have values in the example:

qname	position	CIGAR	lcl	rcl	mcl
align1	3	2M6N8M	2	8	2
align2	2	3M6N7M	3	7	3
align3	1	4M6N3M	4	3	3

2.4.2 Gap-site coordinates

For each gap-site, `localisation-coordinates` are defines as:

- **refid**

- **lend** (left-end) is the (1-based) coordinate of the last matching nucleotide on the left side: `CCCAGGTCCAGCCTTGATGTCC`

- **rstart** (right-start) is the (1-based) coordinate of first matching nucleotide on the right side: `CCCAGGTCCAGCCTTGATGTCC`

We call all aligns that share identical localisation-coordintates `gap-site-defining-aligns`. In order to derive a lower boundary for the size of the adjacent exons are calculated:

- **lstart** (left-start) is the (1-based) coordinate of the leftmost nucleotide for which a match exsists in the set of left adjacent matching regions: `CCCAGGTCCAGCCTTGATGTCC`.
  The position is calculated by $lstart = lend - max(lcl) + 1$.

- **rend** (right-end) is the (1-based) coordinate of the rightmost nucleotide for which a match exists in the set of right adjacent matching regions: `CCCAGGTCCAGCCTTGATGTCC`.
  The position is calculated by $rend = rstart + max(rcl) - 1$.

As derivative, the number of nucleotides in the gap-region (denoted **gaplen**) is calculated as $gaplen = rend - lstart - 1$. Alltogether, the gap-site and the adjacent putative matching regions in this example are:

`CCCAGGTCCAGCCTTGATGCCTTGATGTCC`.

The associated numeric values for the shown example are:

Name	value	base	
refid	0	0	We assume, there is only one reference sequence
lstart	2	1	Leftmost match position (C)
lend	5	1	Last match on left side (G)
rstart	12	1	First match on right side (C)
rend	20	1	Rightmost match position (G)
gaplen	6		Number of nucleotides in gap

2.4.3 Quantification of align numbers

The number of gap-site-defining-aligns are quantified in:

- **nAligns**, the number of aligns that define the gap-site.

- **nProbes**, the number of alignments (BAM-files) in which this gap-site is found.

In the present example, $nAligns = 3$ and $nProbes = 1$.

2.4.4 Quantification of informational support for gap-site's

In order to quantify the information content for each gap-site `lcl` and `mcl` values are stored as single byte values inside of an unsigned long long integer. We define `n` as the number of bytes they contain. On a 32-bit operating system there is

$n = 4$ and on a 64-bit operating system $n = 8$. With that, we van view `lcl` and `mcl` as n-dimensional vectors: $lcl = (lcl_i)_{i=1,\ldots,n}$ and $mcl = (mcl_i)_{i=1,\ldots,n}$ in which values are placed in descending order.

- **nlstart**, the number of different match start positions, which equals the number of different values in the `lcl` vector.
  $nlstart := \#\{lgl_i : i = 1, \ldots, n\}$.
- **lm_sum**, the number of matching nucleotides on the left side of the gap.
  $lm\_sum := \sum_{i=1}^{l} gl_i$.
- **qsm**, the sum of the four largest `mcl` values (quartet sum of minimal cigar length): $\sum_{i=1}^{4} mcl_i$

2.4.5 Gap quality score (gqs)

The gap quality score is calculated as

$$gqs = 10\frac{nlstart}{n} \quad \frac{2qsm}{4} = 10\frac{\#\{lgl_i : i = 1, \ldots, n\}}{n} \quad \frac{2\sum_{i=1}^{4} mcl_i}{4}$$

The score quantifies number of align start positions and matching nucleotides in order to distinguish biological existing splice-sites from alignment phenomenons. The stored information accumulates with increasing the number of included alignments (BAM-files). The score is given as a positive integer value and the maximum reachable number is 10 * read-length.

The higher the score the more likely is the fact that a gap-site represents a splice-site. Be aware that gqs does not quantify gene expression.

3 Object types inside rbamtools package

The description of object types in this section starts with reading and writing access to BAM files, proceeds to objects which elementary data inside BAM files and ends with the description of more complex containers.

3.1 Included example files within rbamtools

There are two example files included which are located in the "/inst/extdata" subdirectory. The directory contains a sorted BAM file "accepted_hits.bam" and the corresponding index file "accepted_hits.bam.bai".
They were produced (using the `extractRanges` function) from a RNA-seq experiment. A human probe was sequenced using an Illumina Hiseq sequencer. Fastq-reads were aligned with tophat against homo sapiens UCSC reference genome. Complex aligns (i.e. nCigar>1) were extracted for genes KLHL17 (chr1) and SNRNP25 (chr16). The BAM file contains 3333 aligns.

3.2 Reading and writing access

Immediate reading and writing access is provided by `bamReader` and `bamWriter` Objects.

3.3 bamReader

An object of class bamReader is constructed and returnd by the function `bamReader` in the following way:

```
> library(rbamtools)
> bam<-system.file("extdata","accepted_hits.bam",package="rbamtools")
> # Open bam file
> reader<-bamReader(bam)
```

An opened bamReader can be used to access the BAM header section and to read aligns sequenitally. `bamReader` can also be used to sort and index BAM files.

Sorting large BAM files requires some time and produces intermediate files. So the recommended way of sorting large BAM files is to use the samtools command line version. Sorting BAM files within R can be done with:

```
> bamSort(reader,prefix="my_sorted",byName=FALSE,maxmem=1e+9)
```

Sorted BAM files can be indexed. Indexing results in a second file which is usually named as the BAM file itself with an added suffix ".bai". An index file can be created with:

```
> create.index(reader,idx_filename="index_file_name.bai")
```

Omitting the `idx_filename` argument results in adding the ".bai" suffix to the filename of the BAM file which is then automatically located in the same directory as the BAM file itself:

```
> create.index(reader)
```

The creation of indexes for large BAM files (10 GB) takes some minutes time but can readily be done with this routine and of course has to be done only once per file.
Index files must be loaded before they can be used:

```
> idx<- system.file("extdata", "accepted_hits.bam.bai", package="rbamtools")
> load.index(reader,idx)
```

The reader object can be checked for for loaded index with:

```
> index.initialized(reader)
```

```
[1] TRUE
```

A shortcut for opening a BAM file and loading the "standard" index at the same time is:

```
> reader<-bamReader(bam,idx=TRUE)
```

3.4 Tabled reference sequences: getRefData

A data.frame with the reference sequences contained in the BAM header can be obtained with:

```
> getRefData(reader)

  ID    SN        LN
1  0  chr1 249250621
2  1 chr16  90354753
```

The returned data.frame contains in the first column (ID) the mentioned refid 2.2.1 value which is not part of the header but uses as identifier for aligns and ranges.

3.5 bamWriter

For creation of a `bamWriter` object, a `bamHeader` and a filename must be given. The most convenient way of obaining a `bamHeader` class is retrieving one from an opened `bamReader` object.

```
> header<-getHeader(reader)
> writer<-bamWriter(header,"test.bam")
> # Write aligns using bamSave
> bamClose(writer)
```

Aligns can be written to a BAM file either from single instances of `bamAlign`'s or from whole `bamRange` objects. Section 1.4

4 Elementary data structures

The content of BAM files can be divided in `header` section and `alignment` section.

4.1 Structures for header section

The complete header information (in binary representation) can be retrieved from a BAM file with the function `getHeader`. An object of this type is needed for creation of a `bamWriter` object. In order to get Access to the data itself, the binary data has to be converted into a string representation which is maintained inside an object of class `bamHeaderText`:

```
> header<-getHeader(reader)
> htxt<-getHeaderText(header)
```

The header section is divided into several seqments (as described above) with data tags that describe the origin of the contained alignments. For each segment there is a class which can be be obtained by calling the appropriate function on a `bamHeaderText` object:

Segment ID	Description	S4 class	Retrieving function
HD	The header line	headerLine	headerLine
SQ	Reference sequence dictionary	refSeqDict	refSeqDict
RG	Read group		
PG	Program	headerProgram	header Program
CO	Comment		

Creating a complete `bamHeader` object from scratch can be done with the following code:

```
> bh<-new("bamHeaderText")
> headl<-new("headerLine")
> setVal(headl,"SO","coordinate")
> dict<-new("refSeqDict")
> addSeq(dict,SN="chr1",LN=249250621)
> addSeq(dict,SN="chr16",LN=90354753)
> dict

An object of class "refSeqDict"
     SN        LN AS M5 SP UR
1  chr1 249250621     0
2 chr16  90354753     0

> prog<-new("headerProgram")
> setVal(prog,"ID","1")
> setVal(prog,"PN","tophat")
> setVal(prog,"CL","tophat --library-type fr-unstranded hs_ucsc_index reads.fastq")
> setVal(prog,"VN","2.0.0")
> bh<-bamHeaderText(head=headl,dict=dict,prog=prog)
> #getHeaderText(bh)
> header<-bamHeader(bh)
```

4.2 Structures for alignment section

Single aligns can be retreaved from opened reader via `getNextAlign`:

```
> align<-getNextAlign(reader)
```

The alignment section in BAM files is a series of alignment (align) records. The data inside of each record is represented by a `bamAlign` object. Section 1.4 [2] describes the information content for each align in detail. The fields and the corresponding `bamAlign` accessors are listed below:

Field	Description	Accessor
QNAME	Name	name
FLAG	Multiple Flags	flag
RNAME	refid	2.2.1 refID
POS	Mapping position	2.2.2 position (0-based!)
MAPQ	Mapping quality	mapQuality
CIGAR	CIGAR string	cigarData
	Number of cigar entries	nCigar
RNEXT	Ref name of mate segment	mateRefID
PNEXT	Position of mate segment	matePosition
SEQ	segment sequence	alignSeq
QUAL	Pred-scaled Quality String	alignQual

The accessors can be used in the following way:

```
> name(align)
> flag(align)
> refID(align)
> position(align)
> mapQuality(align)
> cigarData(align)
> nCigar(align)
> mateRefID(align)
> matePosition(align)
> alignSeq(align)
> alignQual(align)
```

The flag field contains multiple bit-coded flags which are kept together inside an integer value:

Bit	Description	Accessor
0x1	Paired align	paired
0x2	Proper pair	properPair
0x4	Unmapped	unmapped
0x8	Mate umapped	mateUnmapped
0x10	Reverse Strand	reverseStrand
0x20	Mate reverse Strand	mateReverseStrand
0x40	First in pair	firstInPair
0x80	Second in pair	secondInPair
0x100	Secondary align	secondaryAlign
0x200	Not passing quality control	failedQC
0x400	PCR or optical duplicate	`pcrORopt_duplicate`

The following code demonstrates the usage of the flag-accessors:

```
> paired(align)
> properPair(align)
> unmapped(align)
> mateUnmapped(align)
> reverseStrand(align)
> mateReverseStrand(align)
> firstInPair(align)
> secondInPair(align)
> secondaryAlign(align)
> failedQC(align)
> pcrORopt_duplicate(align)
```

The same accessors can also be used to set the accordant values:

```
> unmapped(align)<-TRUE
```

4.2.1 Creating bamAlign objects from scratch

The `bamAlign` function can be used to create *bamAlign* objects from scratch:

```
> align<-bamAlign("HWUSI-0001","ATGTACGTCG","Qual/Strng","4M10N6M",refid=0,position=100)
> align

Class       :    bamAlign
refId       :           0
Position    :         100

Cigar Data  :
  Length Type
0      4    M
1     10    N
2      6    M

> name(align)

[1] "HWUSI-0001"

> alignSeq(align)

[1] "ATGTACGTCG"

> alignQual(align)

[1] "Qual/Strng"

> cigarData(align)

  Length Type
0      4    M
1     10    N
2      6    M

> refID(align)

[1] 0

> position(align)

[1] 100
```

The created bamAlign objects can be added to a *bamRange* list or be written to a BAM-file via *bamWriter*.

5 Complex and cumulative container

5.1 Align lists for specific reference regions: bamRange

bamRange objects manage a list of bamAlign's. As BAM files usually contain alignment results against a reference-genome, bamRange objects contain list of all aligns that match between a given start and stop position on a given chromosome. Region coordinates are thereby defined by a refid 2.2.1 and a start and stop position.

5.1.1 Reading bamRange from bamReader

In order to create a bamRange object, an index-initialized bamReader object and a numeric coordinates-vector of length three are passed to the bamRange function.

There are several ways to provide the coordinates for which the aligns are to be retrieved. The first way is to specify a circumscribed genomic region (e.g. where a gene of interest is located). The names for the coordinates are not required and only added for explanational purposes:

```
> coords<-c(0,899000,900000)
> names(coords)<-c("refid","start","stop")
> range<-bamRange(reader,coords)
> size(range)
```

```
[1] 0
```

The second way is to specify coordinates for a whole reference sequence (chromosome). As can be seen from the output of the getRefData function, the coordinates for the whole first chromosome should be given as:

```
> getRefData(reader)
```

```
  ID    SN        LN
1  0  chr1 249250621
2  1 chr16  90354753
```

```
> coords<-c(0,0,249250621)
> names(coords)<-c("refid","start","stop")
> range<-bamRange(reader,coords)
> size(range)
```

```
[1] 2216
```

The function getRefCoords is used here as shortcut:

```
> coords<-getRefCoords(reader,"chr1")
> coords
```

```
[1]         0         0 249250621
```

```
> range<-bamRange(reader,coords)
> size(range)
```

```
[1] 2216
```

bamRange objects keep a pointer to a current align structure for iteration purposes. Addidionally there are some summarizing values stored (which are displayed by show) which describe the range inside the reference from which the bamRange object was read (seqid,qrBegin,qrEnd,complex) and some statistis (size,qSeqMinLen,qSeqMaxLen). Most of the values are printed upon show:

```
> range
```

```
Class        :    bamRange
Size         :       2.216
Seqid        :           0
qrBegin      :           0
qrEnd        : 249.250.621
Complex      :           0
rSeqLen(LN)  : 249.250.621
qSeqMinLen   :         101
qSeqMaxLen   :         101
Refname      :        chr1
```

```
> getCoords(range)
```

```
    seqid     begin       end
        0         0 249250621
```

```
> getSeqLen(range)
```

```
min max
101 101
```

```
> getParams(range)
```

```
     seqid    qrBegin      qrEnd    complex    rSeqLen qSeqMinLen qSeqMaxLen
         0          0  249250621          0  249250621        101        101
```

```
> getRefName(range)
```

```
[1] "chr1"
```

The (0-based) positions of the leftmost and rightmost matching nucleotides in the align-list are not included by default but can be separately calculated:

```
> getAlignRange(range)
```

```
min_pos max_end
  14398   29867
```

5.1.2 Accessing aligns in bamReader

bamReader objects keep a list of bamAlign objects. The objects can sequentially accessed or a data.frame with the align data can be retrieved. Therefore bamRange objects internally keep a pointer to the current align. When no current align object is set, the next call to getNextAlign will set the current to the first align in list. When the last align in list is reached, the next call to getNextAlign will return NULL.
Sequential access to bamRange objects can be done with getNextAlign:

```
> align<-getNextAlign(range)
```

getNextAlign Sequential access to all contained aligns in a bamRange object can be done with

```
> rewind(range)
> while(!is.null(align))
+ {
+   # Process align data here
+   align<-getNextAlign(range)
+ }
```

A fast way to get tabled align information out of bamRange objects is to use as.data.frame.

```
> rdf<-as.data.frame(range)
```

5.2 gapList

gapList objects represent a list of align gaps. They contain one record for single each align-gap present in align data. Each align-gap can be linked to a single align in the BAM file (via refid and position coordinates).

The function gapList takes an open and indexed instance of bamReader and a set range coordinates (refid,start,stop). The function will scan all aligns that are overlap with the given range in the opened BAM file for gapped aligns. For every contained align gap, the refid and the position of the align, the match length on both sides (left_cigar_len, right_cigar_len) and the (1-based) positions of the last nucleotide the left side of the gap (left_stop) and the (1-based) position of the first nucleotide on the right side of the gap (right_start).

```
> coords<-getRefCoords(reader,"chr1")
> gl<-gapList(reader,coords)
> gl
```

```
An object of class 'gapList'. size: 2297
nAligns: 2216         nAlignGaps: 2297
```

```
> dfr<-as.data.frame(gl)
> dfr[1:6,c(1:3,5:8)]
```

```
  refid position left_cigar_len left_stop gaplen right_start right_cigar_len
0     0    14729            100     14829    140       14970               1
1     0    14729            100     14829    140       14970               1
2     0    14729            100     14829    140       14970               1
3     0    14729            100     14829    140       14970               1
4     0    14729            100     14829    140       14970               1
5     0    14729            100     14829    140       14970               1
```

The columns 4 and 9 contain the type of the adjacent cigar items (which should always be 'M') are omitted.

The size function returns the number of gaps contained in the object. The functions nAligns and nAlignGaps return the total number of aligns and the number of gapped aligns in the scanned range respectively:

```
> size(gl)
> nAligns(gl)
> nAlignGaps(gl)
```

5.3 gapSiteList

`gapSiteList` objects contain pooled align-gap information. The single gaps are condensed by refid, left-stop and right-start. So each combination of coordinates appears only once in the list. The number of aligns in which each gap has been found is counted into the value `nAligns`.

Two `gapSitList` objects can be merged to one. The basic coordinates of the contained gap-sites (refid, lend, rstart) are compared. Gap-sites with no counterpart are just copied into the new list whereas gap-sites with couterpart are merged into one record. In this merging process, the core coordinates are just copied. The following table gives an overview over the calculations which are done for merging:

Column name	Site identificator	Resulting value
id		New running index will be created
refid	+	Copied
lstart		Minimum
lend	+	Copied
rstart	+	Copied
rend		Maximum
gaplen		Copied
nAligns		Sum
nProbes		Sum
nlstart		(See text)
`lm_sum`		(See text)
lcl		(See text)
mcl		(See text)

For `lm_sum`, `lcl` and `mcl`, there are specialiced merging operations.

```
> coords<-getRefCoords(reader,"chr1")
> sl<-siteList(reader,coords)

[gap_site_list_fetch] Fetched list of size 32.

> size(sl)

[1] 32

> nAligns(sl)

[1] 2216

> nAlignGaps(sl)

[1] 2297
```

```
> sl

An object of class 'gapSiteList'. size: 32
nAligns: 2216         nAlignGaps: 2297

> df<-as.data.frame(sl)
> head(df)

  id refid lstart  lend rstart  rend gaplen nAligns nProbes nlstart lm_sum
1  1     0  14730 14829  14970 15052    140     553       1       8    772
2  2     0  14944 15038  15796 15888    757     201       1       8    601
3  3     0  15909 15947  16607 16702    659      29       1       8    196
4  4     0  15953 16027  16607 16669    579       4       1       4    220
5  5     0  16730 16765  16854 16941     88       5       1       5    108
6  6     0  16682 16765  16858 16957     92      34       1       8    358
         lcl       mcl
1 1633837924 842150450
2 1163550303 757935406
3  387456295 387456295
4  640172875 438445608
5  236198180 236198180
6  690630740 690563632
```

5.4 bamGapList

bamGapList Objects are designed to contain information about gap-sites for a complete BAM file (i.e. for all refid's). bamGapList's can be merged, so it's possible to cumulate information about gap-sites from a large number of BAM files (e.g. 50). As the whole collection and merging process is done in C, the whole process usually runs with a processing rate > 1.000.000 aligns/sec (on a desktop machine).

```
> bsl<-bamGapList(reader)
> bsl

An object of class 'bamGapList'. size: 39
nAligns: 3.230         nAlignGaps: 3.443

> size(bsl)

[1] 39

> nAligns(bsl)

[1] 3230

> nAlignGaps(bsl)

[1] 3443

> summary(bsl)
```

```
  ID    SN        LN start size nAligns nAlignGaps
1  0  chr1 249250621     0   32    2216       2297
2  1 chr16  90354753     0    7    1014       1146
```

```
> dfr<-as.data.frame(bsl)
> head(dfr)
```

```
  id seqid lstart  lend rstart  rend gaplen nAligns nProbes nlstart qsm nmcl
0  1  chr1  14730 14829  14970 15052    140     553       1       8 200    8
1  2  chr1  14944 15038  15796 15888    757     201       1       8 181    8
2  3  chr1  15909 15947  16607 16702    659      29       1       8 115    8
3  4  chr1  15953 16027  16607 16669    579       4       1       4 138    4
4  5  chr1  16730 16765  16854 16941     88       5       1       5  95    5
5  6  chr1  16682 16765  16858 16957     92      34       1       8 172    8
   gqs
0 1000
1  905
2  575
3  345
4  296
5  860
```

6 Miscellaneous functions

6.1 bamCount and bamCountAll

The bamCount counts aligns and CIGAR-items in align ranges defined by coordinates. The function returns a named integer vector of length 10.
The bamCountAll counts aligns and CIGAR-items for whole BAM-files (represented by a *bamReader*). The function optionally takes a verbose argument which controls the textual output during runtime. The function returns a *data.frame*. Each line contains counts for one reference sequence, each column contains data for one CIGAR-OP type. Columns with total counts, referene sequence id (ID) and reference sequence length (LN) are added.

```
> bam<-system.file("extdata","accepted_hits.bam",package="rbamtools")
> coords<-c(0,0,14730)
> count<-bamCount(reader,coords)
> count
```

```
      M       I       D       N       S       H       P       =       X nAligns
     30       0       2      13       0       0       0       0       0      15
```

```
> count<-bamCountAll(reader,verbose=TRUE)
```

```
[bamCountAll] Counting chr1        [ 1/2]
[bamCountAll] Counting chr16        [ 2/2]
[bamCountAll] Finished.
```

```
> count
```

```
         M  I  D    N S H P = X nAligns ID        LN
chr1  4577 18 46 2297 0 0 0 0 0    2216  0 249250621
chr16 2164  4  0 1146 0 0 0 0 0    1014  1  90354753
```

6.2 countNucs

The `countNucs` counts occurrence of the nucleotides ACGT in *bamAlign* and *bamRange* objects. An integer vector of length 4 is returned. The names give the nucleotide which is counted at each position. The syntax is identical for *bamAlign*

```
> align<-bamAlign("HWUSI-0001","ACCGGGTTTT","Qual/Strng","4M10N6M",refid=0,position=100)
> countNucs(align)
```

```
A C G T N
1 2 3 4 0
```

and *bamRange*

```
> bam<-system.file("extdata","accepted_hits.bam",package="rbamtools")
> reader<-bamReader(bam,idx=TRUE)
> coords<-c(0,0,14730)
> range<-bamRange(reader,coords)
> countNucs(range)
```

```
  A   C   G   T   N
237 490 533 255   0
```

objects.

6.3 nucStats

nucStats for bamReader The `nucStats` function counts occurrence of the nucleotides ACGT in whole BAM files via opened *bamReader* objects. Any other character values are subsumed in the value N. The last two columns contain values for GC content and AG/GC ratio. The function returns a *data.frame* with one row for each reference sequence which is listed in the BAM-header section.

```
> nucStats(reader)
```

```
      nAligns     A     C     G     T N       gcc at_gc_ratio
chr1     2216 37756 72232 61721 52102 5 0.5835076   0.7137739
chr16    1014 28090 25298 31102 17921 3 0.5835076   0.7137739
```

nucStats for BAM file names The `nucStats` function counts occurrence of the the nucleotides ACGT for a given list of BAM file names. The last two columns contain values for GC content and AG/GC ratio. The function returns a *data.frame* with one row for each given BAM file name.

```
> nucStats(bam)
```

```
  nAligns     A     C     G     T N       gcc at_gc_ratio
1    3230 65846 97530 92823 70023 8 0.5835076   0.7137739
```

6.4 create.idx.batch

The `create.idx.batch` is intended to create index files for a batch of given BAM-files. The names of the created BAM-index files can optionally be added. The standard name for BAM-index files is the name of the BAM file plus an added suffix ".bai". The third (optional) argument is *rebuild*. When *rebuild* is `FALSE` the function will only create not already existing BAM-index files. When *rebuild* is `TRUE` the function will build BAM-index for all given BAM-files.

Sometimes (especially when BAM-files have been copied), they may be errorneous. Rebuilding index files is a way to check the integrity of a BAM-file.

```
> bam<-system.file("extdata","accepted_hits.bam",package="rbamtools")
> create.idx.batch(bam)
```

6.5 reader2fastq, range2fastq

The `reader2fastq` and `range2fastq` take (optionally random subsets) of whole BAM-files (via *bamReader*) or selected ranges (via *bamRange*) and copy aligns to fastq files.
For handling of aligns inside whole BAM-files, use the `reader2fastq` function. Aligns are read from BAM files via `getNextAlign`. For an opened file, there is a pointer to the last retrieved align kept. So multiple calls to `getNextAlign` will retrieve subsequent aligns.
This comes into play when there are precedent calls to `getNextAlign` or a subset has been drawn via a given logical vector. When a logical vector is given, there will be a call to `getNextAlign` for every entry in the vector. The function then returns the number of checked aligns. When EOF is reached before the vector is processed, the number of checked aligns is smaller than the length of the given logical vector. When no logical vector is given, the function returns the number of written aligns.

```
> bam<-system.file("extdata","accepted_hits.bam",package="rbamtools")
> reader<-bamReader(bam)
> reader2fastq(reader,"out.fastq")
> bamClose(reader)
> # Reopen in order to point to first align
> reader<-bamReader(bam)
> index<-sample(1:100,20)
> reader2fastq(reader,"out_subset.fastq",which=index)
```

The function `range2fastq` writes all aligns in a *bamRange* object into a compressed fastq file. Optionally, a logical vector (where length must be equal to size of range) can be given. In this case only the depicted aligns are copied into the fastq file and the remaining alings are skipped.

```
> bam<-system.file("extdata","accepted_hits.bam",package="rbamtools")
> reader<-bamReader(bam,idx=TRUE)
> coords<-as.integer(c(0,0,249250621))
> range<-bamRange(reader,coords)
> range2fastq(range,"rg.fq.gz")
> index<-sample(1:size(range),100)
> range2fastq(range,"rg_subset.fq.gz",which=index)
```

6.6 Functions for reading and displaying Phred quality scores

Phred quality scores Q are defindes as $Q = -10log_{10}P$ where P is the base calling error probability.

getQualDf takes a `bamReader` and returns a data.frame. The data.frame has 94 rows which represent values from 0 to 93 ([1]). The number of columns equals the maximum sequence length in the given `bamRange`.

```
> qdf<-getQualDf(range)
> qdf[32:38,1:10]

   1 2 3 4 5 6 7 8 9 10
31 2 2 1 0 1 0 0 1 1  1
32 0 2 0 3 0 2 0 0 0  0
33 0 0 1 1 3 0 2 1 1  1
34 1 3 1 0 0 0 0 0 0  0
35 0 7 7 7 7 8 7 7 7  6
36 0 0 0 0 0 0 0 0 0  0
37 0 0 0 2 2 2 2 3 0  1

> qdr<-getQualDf(range,prob=TRUE)
> qrr<-round(qdr,2)
> qrr[32:38,1:10]

      1    2    3    4    5    6    7    8    9   10
32 0.13 0.13 0.07 0.00 0.07 0.00 0.00 0.07 0.07 0.07
33 0.00 0.13 0.00 0.20 0.00 0.13 0.00 0.00 0.00 0.00
34 0.00 0.00 0.07 0.07 0.20 0.00 0.13 0.07 0.07 0.07
35 0.07 0.20 0.07 0.00 0.00 0.00 0.00 0.00 0.00 0.00
36 0.00 0.47 0.47 0.47 0.47 0.53 0.47 0.47 0.47 0.40
37 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
38 0.00 0.00 0.00 0.13 0.13 0.13 0.13 0.20 0.00 0.07
```

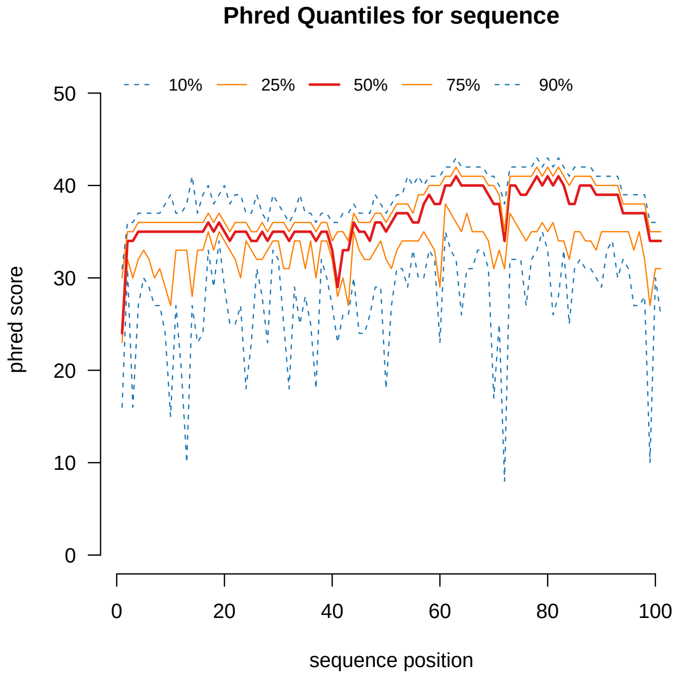
getQualQuantiles takes a `bamReader` and a vector of quantiles (must be between 0 and 1) and returns a data.frame. The data.frame contains one row for each quantile and also as many columns as the maximum sequence length.

```
> qt<-getQualQuantiles(range,c(0.25,0.5,0.75))
> qt[,1:10]

      1  2  3  4  5  6  7  8  9 10
q_25 23 32 30 32 33 32 30 31 29 27
q_50 24 34 34 35 35 35 35 35 35 35
q_75 30 35 35 36 36 36 36 36 36 36
```

plotQualQuant takes a `bamReader` and plots the 0.1, 0.25, 0.5, 0.75 and 0.9 quantiles for all occurring sequence positions.

```
> plotQualQuant(range)
```



6.7 Functions for calculation and displaying align-depth

Align depth means quantification of present matches for each nucleotide position in a given range.

The alignDepth member function calculates align depth for a given bamRange object. From the `bamRange` object, the range is extracted and for each nucleotide position whithin this range the numbers of align matches are calculated. When `alignDepth` is called wich `gap=TRUE`, the function counts aligns solely for gap-adjacent match regions (cigar-op's).

Whe extract a `bamRange` for the WASH7

```
> # WASH7P coordinates
> coords<-as.integer(c(0,16950,17400))
> range<-bamRange(reader,coords)
> bamClose(reader)
> ad<-alignDepth(range)
> ad
```

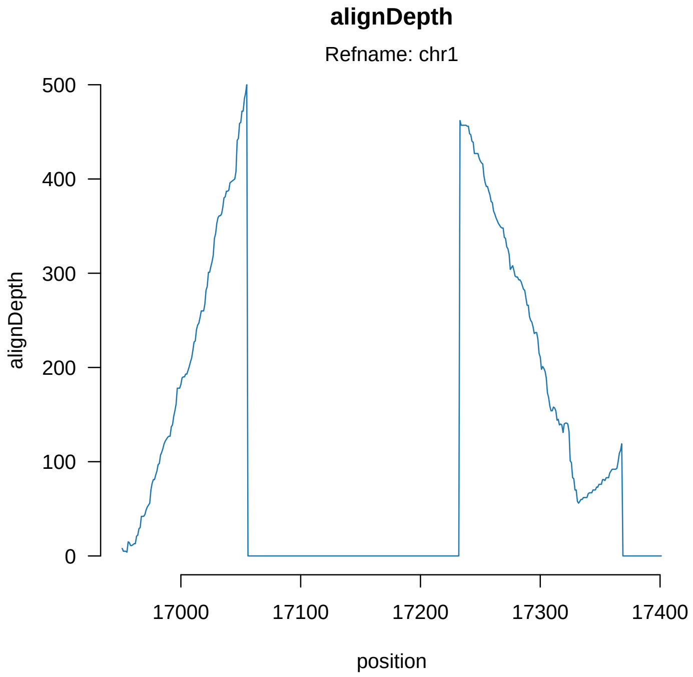
```
Class       :  alignDepth
Seqid       :           0
```

```
qrBegin      :        16.950
qrEnd        :        17.400
Complex      :             0
rSeqLen(LN) :    249.250.621
qSeqMinLen   :           101
qSeqMaxLen   :           101
refname      :          chr1
16951 16952 16953 16954 16955 16956
    8     5     5     5     4    15
```

```
> getParams(ad)
```

```
     seqid    qrBegin      qrEnd    complex     rSeqLen qSeqMinLen qSeqMaxLen
         0      16950      17400          0   249250621        101        101
       gap
         0
```

```
> plotAlignDepth(ad,col="lightblue")
```



References

[1] PJA Cock, CJ Fields, N Goto, ML Heuer, and Rice PM. The sanger fastq file format for sequences with quality scores and the solexa/illumina fastq variants. *Nucleic Acids Research*, 38:1767–1771, 2010.

[2] The SAM Format Specication Working Group. The sam format specication (v1.4-r985). `http://samtools.sourceforge.net/SAM1.pdf`.