

Using spliceSites package

Wolfgang Kaisers, CBiBs HHU Dusseldorf

October 13, 2014

1 Introduction

The data structures and algorithms in this package work on align-gaps which are found in alignments of RNA-seq data. The analysis starts by reading BAM-files [2], so this package assumes that the sequenced RNA is already aligned by external alignment software (e.g. tophat [3]). This package technically builds upon CRAN package 'rBamtools' which performs the reading and data collecting part and CRAN package 'refGenome' from which processed annotation data is imported.

Information on splice events focuses on align-gaps (which are identified by the **N** CIGAR tag) here in this package. Gapped alignments are highly informative because they are calculated by specialized alignment recognition algorithms. Ungapped alignments are only globally counted but not further traced here. This cuts out a relative small but specific part of the alignment information. By doing this possibly valuable information but also many uncertainties are removed from the calculated models.

Align-gaps are assumed to arise from removed introns within the mature mRNA (called splice-sites). This package technically deals with align-gaps but heavily relies on the fact that align-gaps represent exon junctions (EJ). That's why the descriptions contain many switches between the align-view and the EJ view. An important detail at this point is that the inner align frontiers represent exon-intron boundaries. Their positions, read-counts and the surrounding DNA-sequence are the central objectives of the contained algorithms. In contrast, the outer align-boundaries are merely considered as technical artifacts.

Nomenclature. Align-gaps denote gaps in individual aligns. Each align-gap corresponds to a single **N** CIGAR-item. A gap-site is the unique genomic range where align-gaps are placed. Typically, there are many align-gaps which share one gap-site. A gap-site is described by the framing two genomic ranges: The **left** range denotes the one with the lower genomic coordinates which is on the left side of the gap in genome-browser views. The corresponding **right** range denotes the one with the higher genomic coordinates. The **left-right** nomenclature is independent of strand orientation of genes.

Each range is described by a start (left) and an end (right) position. All position values are 1-based, which means that the leftmost character in a sequence

is addressed by 1. Start and stop positions denote the 1-based position of the first and last nucleotide, respectively, which are contained in the range.

The content of one BAM-file is associated here with one biological probe.

2 Formal concepts

2.1 Quantification of gap-site align numbers

Quantification of align numbers for gap-sites differs from the widely used **FPKM** method in that gap sites are not associated with some kind of genomic extend which is addressed by the **K** (kilobase of transcript). Instead number of aligns which contain a specified gap-site (defined by a unique left-end and right-start value) are counted and normalized by a somehow global align number. The spliceSite package provides two quantification indexes.

GPTM *gptm* abbreviates "Gapped Per Ten Million reads". The value represents the relative amount of aligns for a specific EJ in relation to ten million aligned reads per probe. The definition of *gptm* is:

$$gptm = \frac{\text{Number of aligns per gap-site}}{\text{Total number of aligns}} \cdot 10^7.$$

RPMG *rpmg* abbreviates "Reads Per Million Gapped". The value represents the relative amount of aligns for a specific EJ in relation to one million gapped reads per probe. The value is calculated as

$$rpmg = \frac{\text{Number of aligns per splice site}}{\text{Number of gapped aligns in probe}} \cdot 10^6.$$

Both values are influenced by the size of the underlying align pool. During merge operations, the site-specific and total align numbers are summed and the *gptm* and *rpmg* values are recalculated. The resulting values are weighted by the align numbers in each component and differs from the mean. Both values are given as rounded values.

3 Technical description of data structures

Like most software products, this package uses data containers and associated functions. In the following technical part, each container type will be described. The associated functions will be specified direct subsequently for each class.

spliceSites data container The data-structures in this package can be divided in two lineages of data containers and some additional classes which are used for specialized tasks. The two lineages are unilateral container (derived from *cRanges*) and bilateral container (derived from *gapSites*):

- **cRanges** (centered ranges) focus on exon-intron boundaries on one side of the gap-sites. They contain coordinates of genomic ranges (refid,start,end)

and additionally a **position** inside. The **position** points to where the gap-boundary (exon-intron boundary) lies inside the range. The derived classes **cdRanges** and **caRanges** additionally contain DNA and amino acid (AA) sequences.

- **gapSites** (Gapped ranges) focus on two genomic ranges which together surround an gap-sites. The ranges represent two (connected) exons with an interposed intron. The derived classes **annGapSites**, **dnaGapSites** and **aaGapSites** additionally contain annotation data, DNA-sequence and amino-acid sequence respectively.

Additionally there is a class **keyProfiler** which is used to cumulate values for **gapSites** in multiple Probes (BAM-files) for probe subgroups (e.g. gender specific) and a class **maxEntScore** from which maxEntScores can be calculated [4].

3.1 gapSites class lineage

From the base class **gapSites** the lineage derives three child classes: **dnaGapSites**, **aaGapSites** which additionally contain a sequence slot and the class **annGapSites** in which annotation data is shifted into the main table.

gapSites is the central container of the **spliceSite** package. **gapSites** objects are intended to organize information about gap-sites. Typically the collected sites arise from analysis of multiple probes (BAM-files). The underlying concept is to accumulate information about the biological existence of gap-sites over the whole experiment.

The **gapSites** class contains the following slots:

Name	Type	Content
dt	data.frame	Table containing the main (gap-site) data
nAligns	numeric	Total number of aligns counted in the data source
nAlignGaps	numeric	Total number of align gaps counted in the data source
lann	data.frame	Annotation data for left side
rann	data.frame	Annotation data for right side
profile	data.frame	Table describing the probes in the data source

The **nAlignGaps** value counts the number of **N** CIGAR-items in the data source. Therefore aligns with two or more align-gaps generate multiple counts. It is possible albeit uncommon to have more **nAlignGaps** than **nAligns**.

gapSites keep the gap-sites data in the **dt** slot inside a **data.frame**. Each gap-site is represented as one record (line). The table keeps 12 columns which are organized as follows:

Name	Type	Content
id	numeric	Row identifier
seqid	factor	Sequence id (usually chromosome name)
lstart	numeric	Start position of left framing range
lend	numeric	End position left framing range
rstart	numeric	Start position of right framing range
rend	numeric	End position of right framing range
gaplen	numeric	Number of nucleotides in gap
strand	numeric	"+" or "-" or "*"
nAligns	numeric	Number of aligns
nProbes	numeric	Number of probes (BAM-files)
gptm	numeric	Expression quantifier
rpmg	numeric	Expression quantifier

The `gptm` and `rpmg` quantifier contain the previously described quantification scores. To be precise the `lend` value contains the 1-based position of the last exon nucleotide. `rstart` denotes the 1-based position of the first exon nucleotide.

dnaGapSites and aaGapSites Both classes derive from `gapSites` and additionally contain a `seq` slot which contains a `DNAStrngSet` or `AAStringSet` object respectively.

annGapSites `annGapSites` derives from `gapSites` and additionally keeps information about number of probes and annotation data (which is produced by overlapping). `annGapSites` are returned by the member function `annotation` for class `gapSites`.

Creation of gapSites objects is done by directly reading gap-site data from BAM files.

3.2 Reading align data from BAM-files

The `spliceSite` package contains four different functions for reading gap-sites from BAM-files. All of them return `gapSites` objects. `getGapSites` and `alignGapList` read from single BAM files via `bamReader`. `readMergedBamGaps` and `readTabledBamGaps` receive names of BAM-files and return multi-probe merged gap-site data:

Function	Argument	Read range	Profile
<code>getGapSites</code>	<code>bamReader</code>	Range within BAM-file	no
<code>alignGapList</code>	<code>bamReader</code>	BAM-file	no
<code>readMergedBamGaps</code>	filenames	BAM-files	no
<code>readTabledBamGaps</code>	filenames	BAM-files	yes

Existing BAM-indices are an important prerequisite for reading aligns. Either must a given `bamReader` contain an initialized index or a name of BAM-index files must be provided. By default, BAM-index files are expected to be

named as the BAM-files with a ".bai" suffix.

getGapSites reads gap-sites for a given seqid from a single BAM-file (provided as **bamReader**). The seqid argument is given as numeric 1-based index.

```
> library(spliceSites)
> bam<-character(2)
> bam[1]<-system.file("extdata","rna_fem.bam",package="spliceSites")
> bam[2]<-system.file("extdata","rna_mal.bam",package="spliceSites")
> reader<-bamReader(bam[1],idx=TRUE)
> gs<-getGapSites(reader,seqid=1)
> gs
```

Object of class gapSites with 32 rows and 12 columns.

```
nAligns: 2,216      nAlignGaps: 2,297
  id seqid lstart  lend rstart  rend gaplen nAligns strand      gptm
1  1 chr1  14730 14829  14970 15052    140     553      * 2495487.37
2  2 chr1  14944 15038  15796 15888    757     201      * 907039.71
3  3 chr1  15909 15947  16607 16702    659      29      * 130866.43
4  4 chr1  15953 16027  16607 16669    579      4       * 18050.54
5  5 chr1  16730 16765  16854 16941     88      5       * 22563.18
6  6 chr1  16682 16765  16858 16957     92     34       * 153429.60
      rpmg nProbes
1 240748.803      1
2  87505.442      1
3  12625.163      1
4   1741.402      1
5   2176.752      1
6  14801.916      1
```

alignGapList also works on a given **bamReader** but reads gap-sites from the entire file and internally calls **bamGapList**.

```
> ga<-alignGapList(reader)
> ga
```

Object of class gapSites with 46 rows and 16 columns.

```
nAligns: 3,107      nAlignGaps: 3,368
  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 strand      gptm      rpmg
0 1000      * 1779851.95 164192.399
1  905      *  646926.30 59679.335
2  575      *   93337.62  8610.451
```

```

3 345      *   12874.16   1187.648
4 296      *   16092.69   1484.561
5 860      *  109430.32  10095.012

```

Both functions test the given reader for file-open status (via `isOpen`) and for initialized index.

readMergedBamGaps takes a vector of BAM-file names (plus optionally names of the corresponding BAM-index files) and reads gap-site data from each BAM-file (via `rbamtools bamGapList`). gap-site data is merged into a `gapSites` object. The number of files in which each align-gap site is identified is counted in the value `nProbes`.

```

> mbg<-readMergedBamGaps(bam)
> mbg

```

Object of class `gapSites` with 71 rows and 16 columns.

```

nAligns: 7,171      nAlignGaps: 7,665
  id seqid lstart  lend rstart  rend gaplen strand nAligns nProbes      gptm
0  1 chr1  14713 14734  14970 15038    235      *        1        1   1394.506
1  2 chr1  14730 14829  14970 15052    140      *     1126        2 1570213.359
2  3 chr1  14970 15038  15311 15361    272      *        7        1   9761.540
3  4 chr1  14944 15038  15796 15888    757      *     373        2 520150.607
4  5 chr1  15909 15947  16607 16702    659      *       71        2  99009.901
5  6 chr1  15933 16027  16607 16669    579      *        9        2  12550.551
      rpmg nlstart qsm nmcl  gqs
0    130.463      1  22    1  13
1 146901.500      8 200    8 1000
2    913.242      2 103    7  128
3  48662.753      8 182    8  910
4   9262.883      8 121    8  605
5   1174.168      8 170    8  850

```

readTableBamGaps takes a vector of BAM-file names (plus optionally names of the corresponding BAM-index files) and a profile table. The profile table describes the probe profile for each BAM-file (number of BAM-files = number of rows in profile). Every column describes a categorial partition of the BAM-files. For each category, the number of probes (=files), number of aligns and optionally gptm-values are separately calculated. **readTableBamGaps** collects gap-site data (as **readMergedBamGaps** and adds profile columns. The returned `gapSites` object also contains a profile table which can be retrieved via `getProfile`.

```

> prof<-data.frame(gender=c("f","m"))
> rtbg<-readTableBamGaps(bam,prof=prof,rpmg=TRUE)
> rtbg

```

Object of class `gapSites` with 71 rows and 22 columns.

```

nAligns: 7,171      nAlignGaps: 7,665
  id seqid lstart  lend rstart  rend gaplen strand nAligns nProbes nlstart qsm
1  1 chr1  14713 14734  14970 15038    235      *        1        1      1  22

```

2	2	chr1	14730	14829	14970	15052	140	*	1126	2	8	200
3	3	chr1	14970	15038	15311	15361	272	*	7	1	2	103
4	4	chr1	14944	15038	15796	15888	757	*	373	2	8	182
5	5	chr1	15909	15947	16607	16702	659	*	71	2	8	121
6	6	chr1	15933	16027	16607	16669	579	*	9	2	8	170

	nmcl	gqs		gptm		rpmg	c.gender.f	c.gender.m	aln.gender.f
1	1	13		1394.506		130.463	0	1	0
2	8	1000		1570213.359		146901.500	1	1	553
3	7	128		9761.540		913.242	0	1	0
4	8	910		520150.607		48662.753	1	1	201
5	8	605		99009.901		9262.883	1	1	29
6	8	850		12550.551		1174.168	1	1	4

	aln.gender.m	rpmg.gender.f	rpmg.gender.m
1	1	0.000	232.721
2	573	164192.399	133348.848
3	7	0.000	1629.044
4	172	59679.335	40027.926
5	42	8610.451	9774.261
6	5	1187.648	1163.603

```
> getProfile(rtbg)
```

	gender	nAligns	nAlignGaps	nSites	cSites
1	f	3107	3368	46	46
2	m	4064	4297	64	71

infile

```
1 /private/tmp/RtmpXNopD2/Rinsteaf865cf12d4/spliceSites/extdata/rna_fem.bam
2 /private/tmp/RtmpXNopD2/Rinsteaf865cf12d4/spliceSites/extdata/rna_mal.bam
```

3.3 cRanges class lineage

cRanges represent genomic ranges which contain a point of interest inside. In the present setting ranges lie around (left or right) gap-site borders. The class is intended to manage sequence data which crosses exon-intron boundaries. **position** is defined as the 1-based position of the last exon nucleotide. For '+' strand, position=4 means, that usually the 5th and 6th nucleotides are 'GT'.

From the base class **cRanges** the lineage derives two child classes: **cdRanges** and **caRanges** which additionally contain a sequence slot. Sequence information is important for validation splice-sites because required intronic sequence is not contained in the BAM-align structures and must be included from reference sequence.

cRanges contains a **data.frame** in a single slot. Each centered range is represented as one record (line). The table keeps 7 columns which are organized as follows:

Name	Type	Content
seqid	factor	Sequence id (usually chromosome name)
start	numeric	Start position of range
end	numeric	End position range
strand	numeric	"+" or "-" or "*"
position	numeric	
id	numeric	Row identifier
nAligns	numeric	Number of aligns

3.4 Extracting gap-site boundary ranges

xJunc functions extract ranges from **gapSites** objects. **lJunc** and **rJunc** objects extract ranges around one gap-site boundaries and return **cRanges** objects. The position values point to the gap-site (exon-intron) boundary.

rlJunc objects extract ranges from both sides of the gap-site and return **gapSites** objects. Inside the returned object the contained data.frame has two additional columns (**lfeatlen** and **rfeatlen**) which mark the boundary position.

The ranges are intended to be padded with DNA-sequence. Therefore the given strand value is used:

```
> ljp<-lJunc(ga,featlen=6,gaplen=6,strand='+')
> ljp
```

Object of class **cRanges** with 46 rows and 7 columns.

	seqid	start	end	strand	position	id	nAligns
1	chr1	14824	14835	+	6	1	553
2	chr1	15033	15044	+	6	2	201
3	chr1	15942	15953	+	6	3	29
4	chr1	16022	16033	+	6	4	4
5	chr1	16760	16771	+	6	5	5
6	chr1	16760	16771	+	6	6	34

```
> ljm<-lJunc(ga,featlen=6,gaplen=6,strand='-')
> ljm
```

Object of class **cRanges** with 46 rows and 7 columns.

	seqid	start	end	strand	position	id	nAligns
1	chr1	14824	14835	-	6	1	553
2	chr1	15033	15044	-	6	2	201
3	chr1	15942	15953	-	6	3	29
4	chr1	16022	16033	-	6	4	4
5	chr1	16760	16771	-	6	5	5
6	chr1	16760	16771	-	6	6	34

```
> rjp<-rJunc(ga,featlen=6,gaplen=6,strand='+')
> rjp
```


Object of class cRanges with 46 rows and 7 columns.

	seqid	start	end	strand	position	id	nAligns
1	chr1	14964	14975	+	6	1	553
2	chr1	15790	15801	+	6	2	201
3	chr1	16601	16612	+	6	3	29
4	chr1	16601	16612	+	6	4	4
5	chr1	16848	16859	+	6	5	5
6	chr1	16852	16863	+	6	6	34

```
> rjm<-rJunc(ga,featlen=6,gaplen=6,strand='-')
> rjm
```

Object of class cRanges with 46 rows and 7 columns.

	seqid	start	end	strand	position	id	nAligns
1	chr1	14964	14975	-	6	1	553
2	chr1	15790	15801	-	6	2	201
3	chr1	16601	16612	-	6	3	29
4	chr1	16601	16612	-	6	4	4
5	chr1	16848	16859	-	6	5	5
6	chr1	16852	16863	-	6	6	34

```
> lrjp<-lrJunc(ga,lfeatlen=6,rfeatlen=6,strand='+')
> lrjp
```

Object of class gapSites with 46 rows and 14 columns.

nAligns: 3,107		nAlignGaps: 3,368											
	id	seqid	lstart	lend	rstart	rend	gaplen	strand	nAligns	gptm			
1	1	chr1	14824	14829	14970	14975	140	+	553	1779851.95			
2	2	chr1	15033	15038	15796	15801	757	+	201	646926.30			
3	3	chr1	15942	15947	16607	16612	659	+	29	93337.62			
4	4	chr1	16022	16027	16607	16612	579	+	4	12874.16			
5	5	chr1	16760	16765	16854	16859	88	+	5	16092.69			
6	6	chr1	16760	16765	16858	16863	92	+	34	109430.32			
rpmg		nProbes	lfeatlen	rfeatlen									
1	164192.399	1	6	6									
2	59679.335	1	6	6									
3	8610.451	1	6	6									
4	1187.648	1	6	6									
5	1484.561	1	6	6									
6	10095.012	1	6	6									

```
> lrjm<-lrJunc(ga,lfeatlen=6,rfeatlen=6,strand='-')
> lrjm
```

Object of class gapSites with 46 rows and 14 columns.

nAligns: 3,107		nAlignGaps: 3,368											
	id	seqid	lstart	lend	rstart	rend	gaplen	strand	nAligns	gptm			
1	1	chr1	14824	14829	14970	14975	140	-	553	1779851.95			
2	2	chr1	15033	15038	15796	15801	757	-	201	646926.30			
3	3	chr1	15942	15947	16607	16612	659	-	29	93337.62			
4	4	chr1	16022	16027	16607	16612	579	-	4	12874.16			

5	5	chr1	16760	16765	16854	16859	88	-	5	16092.69
6	6	chr1	16760	16765	16858	16863	92	-	34	109430.32
			rpmg	nProbes	lfeatlen	rfeatlen				
1			164192.399	1	6	6				
2			59679.335	1	6	6				
3			8610.451	1	6	6				
4			1187.648	1	6	6				
5			1484.561	1	6	6				
6			10095.012	1	6	6				

xCodons functions. `lCodons` and `rCodons` both take and return `cRanges` objects. The functions provide two tasks:

- Shift start position for sequence extraction according to reading frame.
- Truncate range end to full codon size.

Both operations act unilateral on ranges and so strand information must be provided in order to decide which side of the range (left or right) represents start and end.

`lCodons` and `rCodons` take a numeric frame argument and a logical `keepStrand` argument besides the `cRanges` object. For `frame=2` or `3`, the start position is shifted 1 or 2 nucleotides respectively. The sequence length is then truncated to the largest contained multiple of 3. The functions then correct the position-values in order to keep the positions pointer on the same nucleotide. When `keepStrand` is `FALSE` (the default), the `lCodons` function sets all strand entries to "+" and the `rCodons` sets all Strand entries to "-".

The `lCodons` function should be used for "+"-strand and the `rCodons` function should be used for "-"-strand.

```
> ljp1<-lCodons(ljp,frame=1)
> ljp1
```

Object of class `cRanges` with 46 rows and 8 columns.

	seqid	start	end	strand	position	id	nAligns	frame
1	chr1	14824	14835	+	6	1	553	1
2	chr1	15033	15044	+	6	2	201	1
3	chr1	15942	15953	+	6	3	29	1
4	chr1	16022	16033	+	6	4	4	1
5	chr1	16760	16771	+	6	5	5	1
6	chr1	16760	16771	+	6	6	34	1

```
> ljp2<-lCodons(ljp,frame=2)
> rjm1<-rCodons(ljm,frame=1)
> rjm2<-rCodons(ljm,frame=2)
```

The `xCodons` functions provide a preparative step which allows translation of subsequently added DNA-sequence into AA-sequence. The strand information is therein used to determine the fraction of DNA-sequence must be `reverseComplement`'ed.

The lrCodons function works similar as the xCodons functions but does the same on the two gap-site enframing ranges simultaneously. lrCodons takes and returns gapAligns objects. The strand value can be manually set (default: '*') for later use by dnaGapAligns

```
> lr1<-lrCodons(lrjp,frame=1,strand='+')
> lr2<-lrCodons(lrjp,frame=2,strand='+')
> lr3<-lrCodons(lrjp,frame=3,strand='+')
```

The c-Operator for cRanges and gapSites is used to combine objects made for different frame and strand together to one cRanges or gapSites for joint subsequent analysis:

In the next step we transform these ranges into codons in both directions and all three frames.

```
> ljpg<-c(ljp1,ljp2)
> rjmc<-c(rjm1,rjm2)
> lrj<-c(lr1,lr2,lr3)
```

In order to provide better readable tables there is an optional function for sorting the combined cRanges and gapSites:

```
> ljpg<-sortTable(ljpg)
> rjmc<-sortTable(rjmc)
> lrj<-sortTable(lrj)
```

Trim and resize functions provide upper size limits or fixed size for boundary ranges in gapSites objects. trim_left works on left boundary ranges (i.e. lstart and rend values) and trim_right works on right boundary ranges (i.e. rstart and rend).

All four functions leave the inner boundaries (lend and rstart) unchanged.

```
> trim_left(lrj,3)
> trim_right(lrj,3)
> resize_left(lrj,8)
> resize_right(lrj,8)
```

3.5 Provide additional information

Genuine gapSites and cRanges objects contain numeric coordinate and count data but no sequence or gene association. The conceptual idea is to add gene-annotations and sequence data to primed coordinate containers.

Gene annotation data can be added with the annotate and annotation<- functions. Therefore a refGenome object (ucscGenome or ensemblGenome from package refGenome) must be provided.

```
> ucf<-system.file("extdata","uc_small.RData",package="spliceSites")
> uc<-loadGenome(ucf)
> annotation(ga)<-annotate(ga,uc)
```

Adding annotation data internally works by calling the `overlap` functions (refGenome package). `annotate` calls `overlap` for both framing ranges and adds the results to `gapSites` slots `lann` and `rann`. So there will eventually be different annotation for the two ranges. The same mechanism also works for `gapSites` objects which arise from `readTabledBamGaps`:

Strand information can be deduced from gene annotation. The `getAnnStrand` function looks at the annotation derived strand information on both sides. When the strand information is equal, the function takes the value as strand for the gap-site. Otherwise the strand will be set to '*'. The strand information can be integrated into the internal data.frame by using the `strand` function.

```
> strand(ga)<-getAnnStrand(ga)
```

addGeneAlignPart adds information about distribution of aligns over different gap-sites for each gene.

```
> gap<-addGeneAlignPart(ga)
> gap
```

Object of class `gapSites` with 46 rows and 18 columns.

```
nAligns: 3,107          nAlignGaps: 3,368
```

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

	gqs	strand	gptm	rpmg	gene_aligns	align_part	left_gene_id
1	1000	-	1779851.95	164192.399	NA	NA	uc009viu.3
2	905	-	646926.30	59679.335	234	0.8590	uc009viv.2
3	575	-	93337.62	8610.451	234	0.1239	uc009viv.2
4	345	-	12874.16	1187.648	234	0.0171	uc009viv.2
5	296	-	16092.69	1484.561	NA	NA	uc009viv.2
6	860	-	109430.32	10095.012	NA	NA	uc009viv.2

	left_gene_name	right_gene_id	right_gene_name
1	WASH7P	uc009viv.2	WASH7P
2	WASH7P	uc009viv.2	WASH7P
3	WASH7P	uc009viv.2	WASH7P
4	WASH7P	uc009viv.2	WASH7P
5	WASH7P	uc009viz.2	WASH7P
6	WASH7P	uc009viz.2	WASH7P

DNA-sequence can be added from a `DNAStrngSet` (Biostrings) object via the `dnaRanges` function. `dnaRanges` function takes a `cRanges` object and a `DNAStrngSet` object and adds the corresponding DNA-sequence to each contained range. The function returns an object of class `cdRanges`.

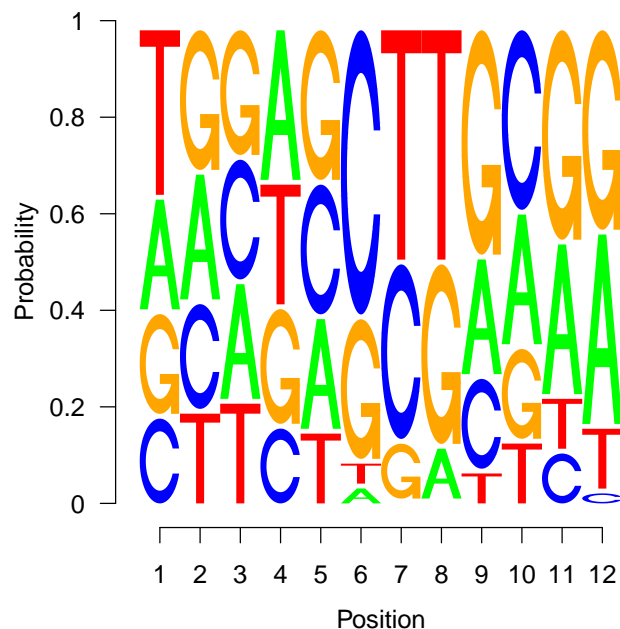
We first load an example `DNAStrngSet` which contains the reference sequence. The sample object `dna_small` contains a small extract of UCSC human reference sequence.

```
> dnafile<-system.file("extdata","dna_small.RData",package="spliceSites")
> load(dnafile)
> dna_small
```

```
A DNASTringSet instance of length 4
      width seq                      names
[1] 104000 NNNNNNNNNNNNNNNNNNNNN...ACTCCGAGGCCATGGACGTGTT chr16
[2]  30000 NNNNNNNNNNNNNNNNNNNNN...AGAAGCAGTGGTCGCCAGGAAT chr1
[3] 360000 NNNNNNNNNNNNNNNNNNNNN...TGGAGCTCAGGGGCACCCACCA chr6
[4] 2851000 NNNNNNNNNNNNNNNNNNNNN...ATCCTCCTGCCTCAGCCTACGC chrY
```

Then we create a `cdRanges` instance `dnaRanges` and then produce a `seqlogo`:

```
> ljpgcd<-dnaRanges(ljpgc,dna_small)
> seqlogo(ljpgcd)
```



Amino-Acid sequences can be obtained after the previous preparatory steps are done. The `translate` function converts a `cdRanges` object into a `caRanges` object by translating the contained DNA-sequences. The function internally uses the Bioconductor Biostrings `translate` function.

```
> ljpgca<-translate(ljpgcd)
```

Extracting subsets from `cRanges` and `gapSites` (and derived classes) can be done with `extractX` functions. They come in two flavours: `extractRange` and `extractByGeneName`. Main application for these function is visual inspection of the results.

`extractRange` takes a `cRanges` or `gapSites` object and a triple of genomic coordinates (`seqid`, `start`, `end`). From the given `cRanges` object, the stored ranges which are contained in the range defined by the coordinates is extracted and returned as `cRanges` object.

```
> # A) For gapSites
> extractRange(ga,seqid="chr1",start=14000,end=30000)
```

Object of class `gapSites` with 32 rows and 16 columns.

```
nAligns: 3,107      nAlignGaps: 3,368
  id seqid lstart  lend rstart  rend gaplen nAligns nProbes nlstart qsm nmcl
1  1 chr1  14730 14829  14970 15052    140    553      1      8 200    8
2  2 chr1  14944 15038  15796 15888    757    201      1      8 181    8
3  3 chr1  15909 15947  16607 16702    659     29      1      8 115    8
4  4 chr1  15953 16027  16607 16669    579      4      1      4 138    4
5  5 chr1  16730 16765  16854 16941     88      5      1      5  95    5
6  6 chr1  16682 16765  16858 16957     92     34      1      8 172    8
  gqs strand      gptm      rpmg left_gene_id left_gene_name right_gene_id
1 1000      - 1779851.95 164192.399 uc009viu.3      WASH7P      uc009viv.2
2  905      -  646926.30  59679.335 uc009viv.2      WASH7P      uc009viv.2
3  575      -  93337.62   8610.451 uc009viv.2      WASH7P      uc009viv.2
4  345      -  12874.16   1187.648 uc009viv.2      WASH7P      uc009viv.2
5  296      -  16092.69   1484.561 uc009viv.2      WASH7P      uc009viz.2
6  860      - 109430.32  10095.012 uc009viv.2      WASH7P      uc009viz.2
  right_gene_name
1      WASH7P
2      WASH7P
3      WASH7P
4      WASH7P
5      WASH7P
6      WASH7P
```

```
> # B) For cRanges
> lj<-lJunc(ga,featlen=3,gaplen=6,strand='+')
> extractRange(lj,seqid="chr1",start=14000,end=30000)
```

Object of class `cRanges` with 32 rows and 7 columns.

```
seqid start  end strand position id nAligns
1 chr1 14827 14835      +      3 1    553
2 chr1 15036 15044      +      3 2    201
3 chr1 15945 15953      +      3 3     29
4 chr1 16025 16033      +      3 4      4
5 chr1 16763 16771      +      3 5      5
6 chr1 16763 16771      +      3 6     34
```

`extractByGeneName` also takes a `cRanges` or `gapSites` object but instead of a numeric range, a vector of gene-names and a `refGenome` object. The `refGenome` object first calculates a set of numerical coordinates from the given gene-names and then calls `extractRange` for each set of coordinates. The resulting objects are then concatenated.

```
> lj<-lJunc(ga,featlen=6,gaplen=3,strand='+')
> ljw<-extractByGeneName(ljp, geneNames="POLR3K",src=uc)
> ljw
```

Object of class cRanges with 2 rows and 7 columns.

	seqid	start	end	strand	position	id	nAligns
33	chr16	97552	97563	+	6 33		168
34	chr16	101640	101651	+	6 34		152

```
> ljw<-extractByGeneName(ljpcd, geneNames="POLR3K",src=uc)
> ljw
```

Object of class cdRanges with 4 rows and 8 columns.

	seqid	start	end	strand	position	id	nAligns	frame	seq
33	chr16	97552	97563	+	6 33		168	1	ACGACTCTGGCA
79	chr16	97553	97561	+	5 33		168	2	CGACTCTGG
34	chr16	101640	101651	+	6 34		152	1	TGTTACCTAACA
80	chr16	101641	101649	+	5 34		152	2	GTTACCTAA

3.6 Working with amino-acid (AA) sequences

For approaching AA based views we first prepare longer DNA sequences for different frames and add DNA-sequence:

```
> l<-12
> lj<-lJunc(mbg,featlen=1,gaplen=1,strand='+')
> lj<-c(lCodons(lj,1),lCodons(lj,2),lCodons(lj,3))
> lrj<-lrJunc(mbg,lfeatlen=1,rfeatlen=1,strand='+')
> lrjc<-c(lrCodons(lrj,1),lrCodons(lrj,2),lrCodons(lrj,3))
> jlrd<-dnaRanges(ljc,dna_small)
```

Translation of DNA sequences can simply be obtained by using translate:

```
> jlrt<-translate(jlrd)
> jlrt
```

Object of class caRanges with 213 rows and 8 columns.

	seqid	start	end	strand	position	id	nAligns	frame	seq
1	chr1	14723	14746	+	5 1		1	1	LCLWLLRW
72	chr1	14724	14744	+	4 1		1	2	SACGCCG
143	chr1	14725	14745	+	4 1		1	3	LPVAAAV
2	chr1	14818	14841	+	5 2		1126	1	SQRCLEGK
73	chr1	14819	14839	+	4 2		1126	2	PRDAWRE
144	chr1	14820	14840	+	4 2		1126	3	PEMPGGK

truncateSeq function addresses stop codons '*' which may be identified in the amino acid sequence. The function truncates the sequence when the stop-codon appears behind (right-hand of) the position and removes rows where the stop-codon appears before (left-hand of) the position. This removes data-sets where the exon-intron boundary lies downstream of a stop-codon.

```
> jlrtt<-truncateSeq(jlrt)
> jlrtt
```

Object of class `caRanges` with 195 rows and 9 columns.

	seqid	start	end	strand	position	id	nAligns	frame	lseq	seq
1	chr1	14723	14746	+	5	1	1	1	8	LCLWLLRW
72	chr1	14724	14744	+	4	1	1	2	7	SACGCCG
143	chr1	14725	14745	+	4	1	1	3	7	LPVAAAV
2	chr1	14818	14841	+	5	2	1126	1	8	SQRCLEGK
73	chr1	14819	14839	+	4	2	1126	2	7	PRDAWRE
144	chr1	14820	14840	+	4	2	1126	3	7	PEMPGGK

trypsinCleave function performs in silico trypsinisation on the provided AA sequences. Trypsin sites are identified by the regex rule "[RK](?!P)" which implements the "Keil"-rule. From the sequence fragments, the one which contains the position marked gap-site (exon-intron) boundary is returned.

```
> jtry<-trypsinCleave(jlrtt)
> jtry<-sortTable(jtry)
```

3.7 Writing output tables

Write functions provide functionality for exporting generated tables into '.csv' files.

write.annDNA.tables writes content of `gapSites` objects together with DNA-sequence segments.

```
> annotation(rtbG)<-annotate(rtbG,uc)
> write.annDNA.tables(rtbG,dna_small,"gapSites.csv",featlen=3,gaplen=8)
```

write.files works on `caRanges` objects and produces two output files: One '.csv' file which contains a copy of the data and a '.fa' file which contains sequence in fasta format.

```
> write.files(jtry,path=getwd(),filename="proteomic",quote=FALSE)
```

The fasta headers contain two tags which are separated by a vertical bar '|'. The first item is the `id` of the corresponding table row and the second item is the prefix of the filename.

3.8 Working with alternative splice-sites

Alternative splice-sites are sites where one exon-intron boundary corresponds to multiple counterparts. In gap-site tables, alternative sites are characterized by multiple occurring entries of `lend` or `rstart`. The `alt_left_ranks` function work by putting gap-sites which share the same `rstart` values in a group which is identified by the same `alt_id` value. Gap-sites which don't share their `rstart` value with any other gap-site have the `alt_id` value 0.

The alt_left_ranks function looks for multiple entries of **rstart** values and returns a table with 6 numeric columns:

Name	Content
id	Row identifier (from gapSites object
alt_id	Group number. Identifies group of gap-sites which share same ; 0 for singular entries
diff_ranks	Rank of gaplen inside same alt_id group
gap_diff	Difference in gaplen to the next greater gaplen value in group
nr_alt	Number of rows with same rstart value

So the rank numbering increases with gaplen (from inside to outside) and the **gap_diff** values are always ≥ 0 because the row with the smallest gaplen gets the rank 1 and gap_diff 0.

When the option **extensive=TRUE** is given, four more columns are contained in the result table: **seq** (seqid), **group** (=rstart, by which the input table is grouped), **alt** (lend) and **len** (gaplen).

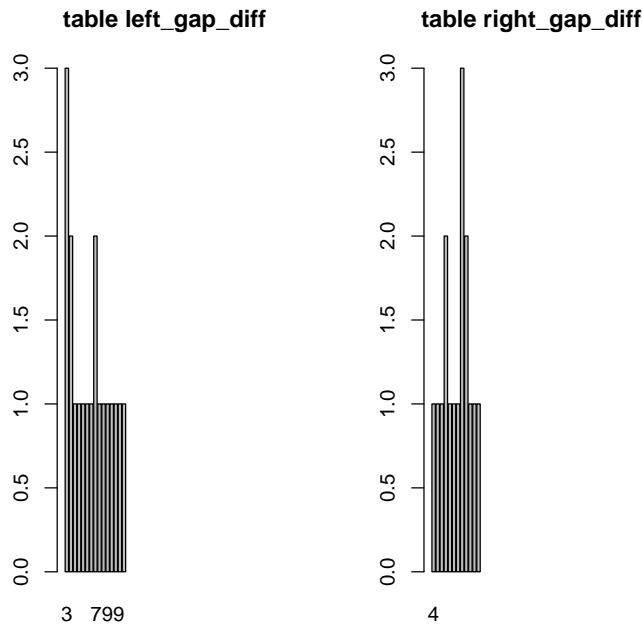
```
> al<-alt_left_ranks(ga)
```

The alt_ranks function combines the results of **alt_left_ranks** and **alt_right_ranks** into one table. There should be characteristic peaks at multiples of 3:

```
> ar<-alt_ranks(ga)
```

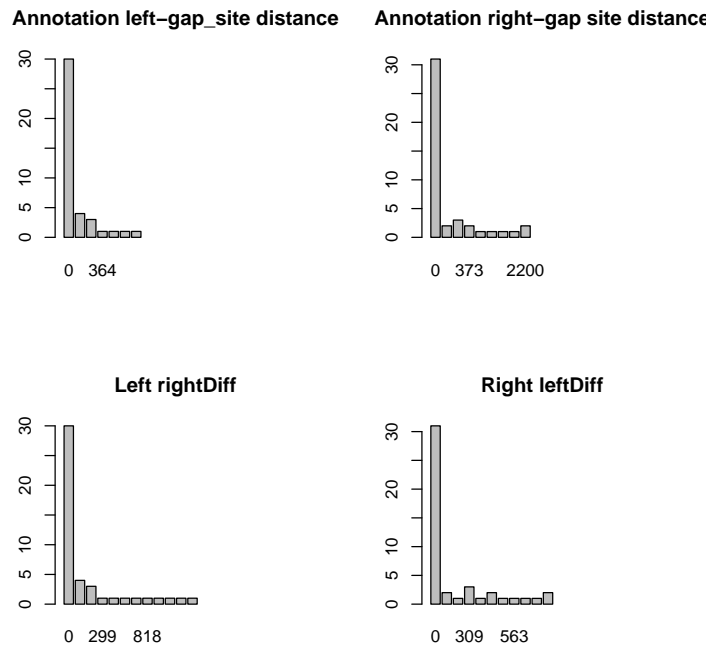
The tabled **gap_diff** values can be plotted with:

```
> plot_diff_ranks(ga)
```



The **plot_diff** function gives a visual overview about the congruence between gap-site positions and the associated annotations. Differences of zero say that a gap-site boundary exactly meets the annotated position.

```
> aga<-annotation(ga)
> plot_diff(aga)
```



MaxEntScores Maximum Entropy (MaxEnt) scores were developed by Gene Yeo and Christopher Burge [4] and are widely accepted as method for quantification of intrinsic EJ strength for a given sequence motif from the exon-intron boundary. MaxEnt scores can be calculated for the 5' and the 3' side of the EJ. MaxEnt scores can be calculated for vectors of sequences for a given position (1-based position of last exon nucleotide) using the **score3** and **score5** functions.

For sliding window calculations on longer DNA-sequences the **scoreSeq5** and **scoreSeq3** functions can be used. The **addMaxEnt** function is used for adding MaxEnt scores to a **gapSites** object.

```
> mes<-load.maxEnt()
> score5(mes,"CCGGTAAGAA",4)

[1] 9.844127

> score3(mes,"CTCTACTACTATCTATCTAGATC",pos=20)

[1] 6.706947
```

```

> sq5<-scoreSeq5(mes,seq="ACGGTAAGTCAGGTAAGT")
> sq3<-scoreSeq3(mes,seq="TTTATTTTCTCACTTTTAGAGACTTCATTCTTTCTCAAATAGGTT")
> gae<-addMaxEnt(ga,dna_small,mes)
> gae

```

Object of class gapSites with 46 rows and 23 columns.

```

nAligns: 3,107      nAlignGaps: 3,368
  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 strand      gptm      rpmg mxe_ps5 mxe_ps3 mxe_ms5 mxe_ms3 s5strand
0 1000      - 1779851.95 164192.399 -30.3    -5.3    6.5    8.3      -
1  905      -  646926.30  59679.335 -21.4   -14.3    9.6    9.2      -
2  575      -  93337.62   8610.451 -26.1   -13.7    6.7    6.5      -
3  345      - 12874.16   1187.648 -17.4   -13.7    6.7    5.8      -
4  296      - 16092.69   1484.561 -19.4   -15.8    8.0    8.7      -
5  860      - 109430.32  10095.012 -19.4   -15.6   10.3    8.7      -
  s3strand meStrand
0      -      -
1      -      -
2      -      -
3      -      -
4      -      -
5      -      -

```

```

> table(getMeStrand(gae))

```

```

+ - *
11 35 0

```

```

> sae<-setMeStrand(gae)
> sae

```

Object of class gapSites with 46 rows and 23 columns.

```

nAligns: 0      nAlignGaps: 0
  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 strand      gptm      rpmg mxe_ps5 mxe_ps3 mxe_ms5 mxe_ms3 s5strand
0 1000      - 1779851.95 164192.399 -30.3    -5.3    6.5    8.3      -
1  905      -  646926.30  59679.335 -21.4   -14.3    9.6    9.2      -
2  575      -  93337.62   8610.451 -26.1   -13.7    6.7    6.5      -
3  345      - 12874.16   1187.648 -17.4   -13.7    6.7    5.8      -

```

4	296	-	16092.69	1484.561	-19.4	-15.8	8.0	8.7	-
5	860	-	109430.32	10095.012	-19.4	-15.6	10.3	8.7	-

	s3strand	meStrand
0	-	-
1	-	-
2	-	-
3	-	-
4	-	-
5	-	-

HBond scores The HBond score provides a measure for the capability of a 5' splice-site to form H-bonds with the U1 snRNA [1]. The HBond score can be calculated for a vector of sequences.

```
> #
> hb<-load.hbond()
> seq<-c("CAGGTGAGTTC","ATGCTGGAGAA","AGGGTGCGGGC","AAGGTAACGTC","AAGGTGAGTTC")
> hbond(hb,seq,3)

[1] 19.4  0.0  8.3 14.1 17.7
```

or can be added to `gapSites` and `cdRanges` objects:

```
> gab<-addHbond(ga,dna_small)
> # D) cdRanges
> lj<-lJunc(ga,featlen=3,gaplen=8,strand='+')
> ljd<-dnaRanges(lj,dna_small)
> ljdh<-addHbond(ljd)
```

3.9 Creating ExpressionSet objects containing gap-site alignment values.

In order to provide technical requirements for analyzing expression data inside the standard Bioconductor framework, there is a `readExpSet` function which produces `ExpressionSet` objects with `rpms` (default) and `gpm` expression values.

readExpSet reads gap-site aligns abundance from a given list of BAM file names into `ExpressionSet`

```
> prof<-data.frame(gender=c("f","m"))
> rtbg<-readTabledBamGaps(bam,prof=prof,rpms=TRUE)
> getProfile(rtbg)

  gender nAligns nAlignGaps nSites cSites
1      f   3107       3368     46     46
2      m   4064       4297     64     71
```



```
infile
1 /private/tmp/RtmpXNopD2/Rinsteaf865cf12d4/spliceSites/extdata/rna_fem.bam
2 /private/tmp/RtmpXNopD2/Rinsteaf865cf12d4/spliceSites/extdata/rna_mal.bam
```

```
> meta<-data.frame(labelDescription=names(prof),row.names=names(prof))
> pd<-new("AnnotatedDataFrame",data=prof,varMetadata=meta)
> es<-readExpSet(bam,phenoData=pd)
```

There are two annotation functions for ExpressionSets which are created by readExpSet: `annotate` and `uniqueJuncAnn`. `Annotate` finds overlaps to a given `refGenome` object. `uniqueJuncAnn` finds exact matches with known splice-sites:

```
> ann<-annotate(es,uc)
> ucj<-getSpliceTable(uc)
> uja<-uniqueJuncAnn(es,ucj)
```

readCuffGeneFpkm reads FPKM values from all given cufflinks files and collects the values into an **ExpressionSet**. In order to get unique gene identifier, the contained values are grouped and for each gene the maximum FPKM values is selected.

```
> n<-10
> cuff<-system.file("extdata","cuff_files",paste(1:n,"genes","fpkm_tracking",sep="."),pack
> gr<-system.file("extdata","cuff_files","groups.csv",package="spliceSites")
> groups<-read.table(gr,sep="\t",header=TRUE)
> meta<-data.frame(labelDescription=c("gender","age-group","location"),row.names=c("gen","
> phenoData<-new("AnnotatedDataFrame",data=groups,varMetadata=meta)
> exset<-readCuffGeneFpkm(cuff,phenoData)
```

4 Appendix

4.1 The keytable class

The **keytable** class is designed to count align data and for computation of expression values for experimental groups separately. The class is only internally used by the **readTabledBamGaps** function.

Group assignment of probes (BAM-files) is done by providing a profile table. Align count values then can subsequently be added to the created object. After data-collection a result table can be retrieved.

For description of functionality first artificial data is created. A profile-table defines the group association of all analyzed probes (usually each probe is one BAM-file).

```
> prof<-data.frame(gen=factor(c("w","m","w","w"),levels=c("m","w")),
+                  loc=factor(c("thx","thx","abd","abd"),levels=c("thx","abd")),
+                  ag =factor(c("y","y","m","o"),levels=c("y","m","o")))
> prof
```

	gen	loc	ag
1	w	thx	y
2	m	thx	y
3	w	abd	m
4	w	abd	o

We the create artificial align-count data for several some gap-sites. The input table `key` allows for multipe entries for the same site. For the output the sites are merged (and align numbers summed) into `ku`.

```
> key1<-data.frame(id=1:5,
+                  seqid=c(1,1,2,2,3),
+                  lend=c(10,20,10,30,10),
+                  rstart=c(20,30,20,40,20),
+                  nAligns=c(11,21,31,41,51))
> key2<-data.frame(id=1:5,
+                  seqid=c(1,1,2,2,4),
+                  lend=c(10,20,10,30,50),
+                  rstart=c(20,30,20,40,70),
+                  nAligns=c(21,22,23,24,25))
> key3<-data.frame(id=1:5,
+                  seqid=c(1,2,4,5,5),
+                  lend=c(10,10,60,10,20),
+                  rstart=c(20,20,80,20,30),
+                  nAligns=c(31,32,33,34,35))
> key<-rbind(key1,key2,key3)
> # Group positions
> ku<-aggregate(data.frame(nAligns=key$nAligns),
+               by=list(seqid=key$seqid,lend=key$lend,rstart=key$rstart),
+               FUN=sum)
```

The next steps comes in two versions: one version where only number of probes are counted for each site and the second version where aligns are counted for each site. The first example shows the probe-counting procedure: The `keyProfiler` object is created from the first probe data and subsequently data for two probes is added (via `addKeyTable`):

```
> # Count probes
> kpc<-new("keyProfiler",keyTable=key1[,c("seqid","lend","rstart")],prof=prof)
> addKeyTable(kpc,keyTable=key2[,c("seqid","lend","rstart")],index=2)
> addKeyTable(kpc,keyTable=key3[,c("seqid","lend","rstart")],index=4)
>
```

The result is then appended to the grouped input table:

```
> cp<-appendKeyTable(kpc,ku,prefix="c.")
> cp
```

	seqid	lend	rstart	nAligns	c.gen.m	c.gen.w	c.loc.thx	c.loc.abd	c.ag.y	c.ag.m
1	1	10	20	63	1	2	2	1	2	0
2	1	20	30	43	1	1	2	0	2	0
3	2	10	20	86	1	2	2	1	2	0
4	2	30	40	65	1	1	2	0	2	0
5	3	10	20	51	0	1	1	0	1	0
6	4	50	70	25	1	0	1	0	1	0
7	4	60	80	33	0	1	0	1	0	0
8	5	10	20	34	0	1	0	1	0	0
9	5	20	30	35	0	1	0	1	0	0

```

      c.ag.o
1      1
2      0
3      1
4      0
5      0
6      0
7      1
8      1
9      1

```

The second version counts align numbers over probes:

```

> # Count aligns
> kpa<-new("keyProfiler",keyTable=key1[,c("seqid","lend","rstart")],prof=prof,values=key1$
> kpa@ev$dtb

      seqid lend rstart gen.m gen.w loc.thx loc.abd ag.y ag.m ag.o
1      1   10    20      0   11    11      0   11   0   0
2      1   20    30      0   21    21      0   21   0   0
3      2   10    20      0   31    31      0   31   0   0
4      2   30    40      0   41    41      0   41   0   0
5      3   10    20      0   51    51      0   51   0   0

> addKeyTable(kpa,keyTable=key2[,c("seqid","lend","rstart")],index=2,values=key2$nAligns)
> addKeyTable(kpa,keyTable=key3[,c("seqid","lend","rstart")],index=4,values=key3$nAligns)
> ca<-appendKeyTable(kpa,ku,prefix="aln.")
> ca

      seqid lend rstart nAligns aln.gen.m aln.gen.w aln.loc.thx aln.loc.abd
1      1   10    20      63      21      42      32      31
2      1   20    30      43      22      21      43      0
3      2   10    20      86      23      63      54      32
4      2   30    40      65      24      41      65      0
5      3   10    20      51       0      51      51      0
6      4   50    70      25      25       0      25      0
7      4   60    80      33       0      33       0      33
8      5   10    20      34       0      34       0      34
9      5   20    30      35       0      35       0      35

      aln.ag.y aln.ag.m aln.ag.o
1      32      0      31
2      43      0      0
3      54      0      32
4      65      0      0
5      51      0      0
6      25      0      0
7       0      0      33
8       0      0      34
9       0      0      35

```

The `readTableBamGaps` function uses both versions simultaneously: Two `keyProfiler` objects keep probe and align data separately. The two tables are then appended to the key table subsequently.

References

- [1] M Freund, C Asang, S Kammler, C Konermann, J Krummheuer, M Hipp, I Meyer, W Gierling, S Theiss, T Preuss, D Schindler, K Jems J, and H Schaal. A novel approach to describe a u1 snrna binding site. *Nucleic Acids Research*, 31:6963–6975, 2003. http://www.uni-duesseldorf.de/rna/html/hbond_score.php.
- [2] The SAM Format Specification Working Group. The sam format specification (v1.4-r985). <http://samtools.sourceforge.net/SAM1.pdf>.
- [3] D Kim, C Pertea, C Trapnell, H Pimentel, R Kelley, and SL Salzberg. Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, 14:R36, 2013. <http://tophat.cbcb.umd.edu/>.
- [4] G Yeo and CB Burge. Maximum entropy modeling of short sequence motifs with applications to rna splicing signals. *J Comput Biol*, 11:377–394, 2004. http://genes.mit.edu/burgelab/maxent/Xmaxentscan_scoreseq.html.