GenomicTuples: Classes and Methods

Peter Hickey*

Modified: 09 October, 2014. Compiled: October 13, 2014

Contents

1 Introduction **2**
1.1 What is a genomic tuple? . . . 2
1.2 When would you need a genomic tuple? . . . 2

2 GTuples **2**
2.1 *GTuples* methods . . . 4
2.1.1 Basic *GTuples* accessors . . . 4
2.1.2 Splitting and combining *GTuples* objects . . . 5
2.1.3 Subsetting *GTuples* objects . . . 6
2.1.4 Basic tuple operations for *GTuples* objects . . . 8
2.1.5 Additional methods unique to *GTuples* . . . 10
2.2 Implementation details . . . 11

3 GTuplesList **11**
3.1 *GTuplesList* methods . . . 12
3.1.1 Basic *GTuplesList* accessors . . . 12
3.1.2 Combining *GTuplesList* objects . . . 14
3.1.3 Subsetting *GTuplesList* objects . . . 14
3.1.4 Basic tuple operations for *GTuplesList* objects . . . 18
3.1.5 Looping over *GTuplesList* objects . . . 20
3.1.6 Additional methods unique to *GTuplesList* . . . 23
3.2 Implementation details . . . 23

4 `findOverlaps`-based methods **24**
4.1 Definition of overlapping genomic tuples . . . 24
4.2 Definition of overlapping genomic tuples and ranges . . . 24
4.3 Examples . . . 24

5 Comparison of genomic tuples **31**
5.1 Definition of comparison methods for tuples . . . 31
5.2 Examples . . . 31

6 Acknowledgements **33**

7 Session info **34**

*peter.hickey@gmail.com

1 Introduction

The *GenomicTuples* *R* package defines general purpose containers for storing *genomic tuples*. It aims to provide functionality for tuples of genomic co-ordinates that are analogous to those available for genomic ranges in the *GenomicRanges* *Bioconductor* package.

As you will see, the functionality of the *GenomicTuples* package is based almost entirely on the wonderful *GenomicRanges* package. Therefore, I have tried to keep the user interface as similar as possible. This vignette is also heavily based on the vignette "An Introduction to Genomic Ranges Classes", which is included with the *GenomicRanges* package.

comment: While not essential, familiarity with the GenomicRanges will be of benifit in understanding the GenomicTuples package.

1.1 What is a genomic tuple?

A genomic tuple is defined by a *sequence name* (`seqnames`), a *strand* (`strand`) and a tuple (`tuples`). All positions in a genomic tuple must be on the same strand and sorted in ascending order. Each tuple has an associated `size`. For example, `chr1:+:{34, 39, 60}` is a 3-tuple (`size` $= 3$) of the positions `chr1:34`, `chr1:39` and `chr1:60` on the + strand. The `size` of a tuple is an integer, $1, 2, \ldots$.

When referring to genomic tuples of a general (fixed) `size`, I will abbreviate these to *m*-tuples, where $m =$ `size`. I will refer to the first position as pos_1 (`pos1`), the second as pos_2 (`pos2`), $\ldots$, and the final position as pos_m (`posm`).

The difference between a genomic tuple and a genomic range can be thought of as the difference between a set and an interval. For example, the genomic tuple `chr10:-:{800, 900}` only includes the positions `chr10:-:800` and `chr10:-:900` whereas the genomic range `chr10:-:[800, 900]` includes the positions `chr10:-:800`, `chr10:-:801`, `chr10:-:802`, $\ldots$, `chr10:-:900`.

1.2 When would you need a genomic tuple?

In short, whenever the co-ordinates of your genomic data are better defined by a set than by an interval.

The original use case for the *GTuples* class was to store the genomic co-ordinates of "methylation patterns". I am currently developing these ideas in a separate R package, *MethylationTuples*, which makes heavy use of the *GTuples* class. Other genomic data, such as long reads containing multiple variants, may also be better conceptualised as genomic tuples rather than as genomic ranges and therefore may benefit from the *GenomicTuples* infrastructure.

2 GTuples

The *GTuples* class represents a collection of genomic tuples, where each tuple has the same `size`. These objects can be created by using the `GTuples` constructor function. For example,

```
library(GenomicTuples)
```

```
## Loading required package:  GenomicRanges
## Loading required package:  BiocGenerics
## Loading required package:  parallel
##
## Attaching package:  'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ, clusterExport,
```

```
##    clusterMap, parApply, parCapply, parLapply, parLapplyLB, parRapply,
##    parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##    xtabs
##
## The following objects are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, append, as.data.frame,
##    as.vector, cbind, colnames, do.call, duplicated, eval, evalq, get,
##    intersect, is.unsorted, lapply, mapply, match, mget, order, paste, pmax,
##    pmax.int, pmin, pmin.int, rank, rbind, rep.int, rownames, sapply, setdiff,
##    sort, table, tapply, union, unique, unlist, unsplit
##
## Loading required package:  S4Vectors
## Loading required package:  stats4
## Loading required package:  IRanges
## Loading required package:  GenomeInfoDb
```

```
seqinfo <- Seqinfo(paste0("chr", 1:3), c(1000, 2000, 1500), NA, "mock1")
gt3 <- GTuples(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"),
                              c(1, 3, 2, 4)),
               tuples = matrix(c(1:10, 2:11, 3:12), ncol = 3),
               strand = Rle(strand(c("-", "+", "*", "+", "-")),
                            c(1, 2, 2, 3, 2)),
               score = 1:10, GC = seq(1, 0, length = 10), seqinfo = seqinfo)
names(gt3) <- letters[1:10]
gt3
```

```
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   a     chr1    1    2    3      - |     1                 1
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   c     chr2    3    4    5      + |     3 0.777777777777778
##   d     chr2    4    5    6      * |     4 0.666666666666667
##   e     chr1    5    6    7      * |     5 0.555555555555556
##   f     chr1    6    7    8      + |     6 0.444444444444444
##   g     chr3    7    8    9      + |     7 0.333333333333333
##   h     chr3    8    9   10      + |     8 0.222222222222222
##   i     chr3    9   10   11      - |     9 0.111111111111111
##   j     chr3   10   11   12      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

creates a *GTuples* object with 10 genomic tuples. The output of the *GTuples* show method is very similar to that of the show method for *GenomicRanges::GRanges* objects. Namely, it separates the information into a left and right hand region that are separated by | symbols. The genomic coordinates (`seqnames`, `tuples`, and `strand`) are located on the left-hand side and the metadata columns (annotation) are located on the right. For this example, the metadata is comprised of `score` and `GC` information, but almost anything can be stored in the metadata portion of a *GTuples* object.

comment: The main difference between a GTuples object and GenomicRanges::GRanges object is that the former uses `tuples` while the latter uses `ranges` in the genomic coordinates.

For even more information on the *GTuples* class be sure to consult the manual page:

```
?GTuples
```

2.1 GTuples methods

Most methods defined for *GenomicRanges::GRanges* are also defined for *GTuples*. Those that are not yet defined, which are those that make sense for ranges but generally not for tuples, return error messages.

If you require a method that is not defined for *GTuples* but is defined for *GenomicRanges::GRanges*, then this can be achieved by first coercing the *GTuples* object to a *GenomicRanges::GRanges* object; *warning: coercing a GTuples object to a GenomicRanges::GRanges is generally a destructive operation*.

```
as(gt3, "GRanges")
```

```
## GRanges object with 10 ranges and 0 metadata columns:
##     seqnames    ranges strand
##        <Rle> <IRanges>  <Rle>
##   a     chr1  [ 1,  3]      -
##   b     chr2  [ 2,  4]      +
##   c     chr2  [ 3,  5]      +
##   d     chr2  [ 4,  6]      *
##   e     chr1  [ 5,  7]      *
##   f     chr1  [ 6,  8]      +
##   g     chr3  [ 7,  9]      +
##   h     chr3  [ 8, 10]      +
##   i     chr3  [ 9, 11]      -
##   j     chr3  [10, 12]      -
##   -------
##   seqinfo: 3 sequences from mock1 genome
```

2.1.1 Basic GTuples accessors

The components of the genomic coordinates within a *GTuples* object can be extracted using the `seqnames`, `tuples` (see Section), and `strand` accessor functions. *warning: The `tuples` accessor should be used in place of the `ranges` accessor. While the `ranges` method is well-defined, namely it accesses pos_1 and pos_m of the object, this is not generally what is desired or required.*

```
seqnames(gt3)
```

```
## factor-Rle of length 10 with 4 runs
##   Lengths:    1    3    2    4
##   Values : chr1 chr2 chr1 chr3
## Levels(3): chr1 chr2 chr3
```

```
tuples(gt3)
```

```
##       pos1 pos2 pos3
##  [1,]    1    2    3
##  [2,]    2    3    4
##  [3,]    3    4    5
##  [4,]    4    5    6
##  [5,]    5    6    7
##  [6,]    6    7    8
##  [7,]    7    8    9
##  [8,]    8    9   10
##  [9,]    9   10   11
```

```
## [10,]   10   11   12
```

```
strand(gt3)
```

```
## factor-Rle of length 10 with 5 runs
##   Lengths: 1 2 2 3 2
##   Values : - + * + -
## Levels(3): + - *
```

Stored annotations for these coordinates can be extracted as a `DataFrame` object using the `mcols` accessor:

```
mcols(gt3)
```

```
## DataFrame with 10 rows and 2 columns
##        score        GC
##    <integer> <numeric>
## 1          1 1.0000000
## 2          2 0.8888889
## 3          3 0.7777778
## 4          4 0.6666667
## 5          5 0.5555556
## 6          6 0.4444444
## 7          7 0.3333333
## 8          8 0.2222222
## 9          9 0.1111111
## 10        10 0.0000000
```

`Seqinfo` can be extracted using the `seqinfo` accessor:

```
seqinfo(gt3)
```

```
## Seqinfo object with 3 sequences from mock1 genome:
##   seqnames seqlengths isCircular genome
##   chr1           1000         NA  mock1
##   chr2           2000         NA  mock1
##   chr3           1500         NA  mock1
```

Methods for accessing the length and names have also been defined:

```
length(gt3)
```

```
## [1] 10
```

```
names(gt3)
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

2.1.2 Splitting and combining GTuples objects

GTuples objects can be devided into groups using the `split` method. This produces a *GTuplesList* object, a class that will be discussed in detail in the next section:

```
sp <- split(gt3, rep(1:2, each=5))
sp
```

```
## GTuplesList object of length 2:
## $1
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   a     chr1    1    2    3      - |     1                 1
```

```
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   c     chr2    3    4    5      + |     3 0.777777777777778
##   d     chr2    4    5    6      * |     4 0.666666666666667
##   e     chr1    5    6    7      * |     5 0.555555555555556
##
## $2
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   f     chr1    6    7    8      + |     6 0.444444444444444
##   g     chr3    7    8    9      + |     7 0.333333333333333
##   h     chr3    8    9   10      + |     8 0.222222222222222
##   i     chr3    9   10   11      - |     9 0.111111111111111
##   j     chr3   10   11   12      - |    10                 0
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

If you then grab the components of this list, they can also be merged by using the c and append methods:

```
c(sp[[1]], sp[[2]])
```

```
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   a     chr1    1    2    3      - |     1                 1
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   c     chr2    3    4    5      + |     3 0.777777777777778
##   d     chr2    4    5    6      * |     4 0.666666666666667
##   e     chr1    5    6    7      * |     5 0.555555555555556
##   f     chr1    6    7    8      + |     6 0.444444444444444
##   g     chr3    7    8    9      + |     7 0.333333333333333
##   h     chr3    8    9   10      + |     8 0.222222222222222
##   i     chr3    9   10   11      - |     9 0.111111111111111
##   j     chr3   10   11   12      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

2.1.3 Subsetting GTuples objects

The expected subsetting operations are also available for *GTuples* objects:

```
gt3[2:3]
```

```
## GTuples object with 2 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   c     chr2    3    4    5      + |     3 0.777777777777778
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

A second argument to the [subset operator can be used to specify which metadata columns to extract from the *GTuples* object. For example:

```
gt3[2:3, "GC"]
```

```
## GTuples object with 2 x 3-tuples and 1 metadata column:
##     seqnames pos1 pos2 pos3 strand |                GC
```

```
##   b     chr2    2    3    4      + | 0.888888888888889
##   c     chr2    3    4    5      + | 0.777777777777778
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

You can also assign into elements of the *GTuples* object. Here is an example where the 2nd row of a *GTuples* object is replaced with the 1st row of gt3:

```
gt3_mod <- gt3
gt3_mod[2] <- gt3[1]
head(gt3_mod, n = 3)
```

```
## GTuples object with 3 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   a     chr1    1    2    3      - |     1                 1
##   b     chr1    1    2    3      - |     1                 1
##   c     chr2    3    4    5      + |     3 0.777777777777778
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

There are also methods to repeat, reverse, or select specific portions of *GTuples* objects:

```
rep(gt3[2], times = 3)
```

```
## GTuples object with 3 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
rev(gt3)
```

```
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   j     chr3   10   11   12      - |    10                 0
##   i     chr3    9   10   11      - |     9 0.111111111111111
##   h     chr3    8    9   10      + |     8 0.222222222222222
##   g     chr3    7    8    9      + |     7 0.333333333333333
##   f     chr1    6    7    8      + |     6 0.444444444444444
##   e     chr1    5    6    7      * |     5 0.555555555555556
##   d     chr2    4    5    6      * |     4 0.666666666666667
##   c     chr2    3    4    5      + |     3 0.777777777777778
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   a     chr1    1    2    3      - |     1                 1
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
head(gt3, n = 2)
```

```
## GTuples object with 2 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   a     chr1    1    2    3      - |     1                 1
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
tail(gt3, n = 2)
```

```
## GTuples object with 2 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   i     chr3    9   10   11      - |     9 0.111111111111111
##   j     chr3   10   11   12      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
window(gt3, start = 2, end = 4)
```

```
## GTuples object with 3 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   b     chr2    2    3    4      + |     2 0.888888888888889
##   c     chr2    3    4    5      + |     3 0.777777777777778
##   d     chr2    4    5    6      * |     4 0.666666666666667
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

2.1.4 Basic tuple operations for GTuples objects

Basic tuple characteristics of *GTuples* objects can be extracted using the `start`, `end`, and `tuples` methods. *warning: While the `width` method is well-defined, namely as $pos_m - pos_1 + 1$, this may not be what is required. Instead, please see the `IPD` method that will be discussed in the next section*.

```
start(gt3)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
end(gt3)
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12
```

```
tuples(gt3)
```

```
##       pos1 pos2 pos3
##  [1,]    1    2    3
##  [2,]    2    3    4
##  [3,]    3    4    5
##  [4,]    4    5    6
##  [5,]    5    6    7
##  [6,]    6    7    8
##  [7,]    7    8    9
##  [8,]    8    9   10
##  [9,]    9   10   11
## [10,]   10   11   12
```

Intra-tuple operations Most of the intra-range methods defined for *GenomicRanges::GRanges* objects are not currently defined via extension for *GTuples* objects due to the differences between ranges and tuples. Those not currently defined, and which return an error message, are:

- `narrow`
- `flank`
- `promoters`
- `resize`
- `Ops`

I am happy to add these methods if appropriate, so please contact me if you have suggestions for good definitions.

Both the `trim` and `shift` methods are well-defined, although the former is somewhat limited since it will return an error if the *internal positions* exceed the `seqlengths`:

```
shift(gt3, 500)

## GTuples object with 10 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   a     chr1  501  502  503      - |     1                 1
##   b     chr2  502  503  504      + |     2 0.888888888888889
##   c     chr2  503  504  505      + |     3 0.777777777777778
##   d     chr2  504  505  506      * |     4 0.666666666666667
##   e     chr1  505  506  507      * |     5 0.555555555555556
##   f     chr1  506  507  508      + |     6 0.444444444444444
##   g     chr3  507  508  509      + |     7 0.333333333333333
##   h     chr3  508  509  510      + |     8 0.222222222222222
##   i     chr3  509  510  511      - |     9 0.111111111111111
##   j     chr3  510  511  512      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
# Raises warning due to tuple being outside of seqlength
x <- shift(gt3[1], 999)

## Warning in valid.GenomicRanges.seqinfo(x, suggest.trim = TRUE): GRanges object contains 1 out-of-bound
range located on sequence chr1.  Note that
##  only ranges located on a non-circular sequence whose length is not NA can be
##  considered out-of-bound (use seqlengths() and isCircular() to get the lengths
##  and circularity flags of the underlying sequences).  You can use trim() to trim
##  these ranges.  See ¿trim,GenomicRanges-method‘ for more information.

## Warning in valid.GenomicRanges.seqinfo(x, suggest.trim = TRUE): GRanges object contains 1 out-of-bound
range located on sequence chr1.  Note that
##  only ranges located on a non-circular sequence whose length is not NA can be
##  considered out-of-bound (use seqlengths() and isCircular() to get the lengths
##  and circularity flags of the underlying sequences).  You can use trim() to trim
##  these ranges.  See ¿trim,GenomicRanges-method‘ for more information.

x

## GTuples object with 1 x 3-tuple and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score GC
##   a     chr1 1000 1001 1002      - |     1  1
##   ---
##   seqinfo: 3 sequences from mock1 genome

# Returns an error because internal position exceeds sequence length, resulting
# in a malformed tuple when trimmed.
trim(x)

## Error in validObject(object):  invalid class "GTuples" object:  positions in each tuple must be
sorted in strictly increasing order, i.e.  'pos1' < ...  < 'pos3'
```

Inter-tuple operations None of the inter-range methods defined for *GenomicRanges::GRanges* objects are currently defined via extension for *GTuples* objects due to the differences between ranges and tuples. Those not currently defined, and which return an error message, are:

- `range`
- `reduce`

- `gaps`
- `disjoin`
- `isDisjoint`
- `disjointBins`

I am happy to add these methods if appropriate, so please contact me if you have suggestions for good definitions.

Interval set operations for GTuples objects None of the interval set operations defined for *GenomicRanges::GRanges* objects are currently defined via extension for *GTuples* objects due to the differences between ranges and tuples. Those not currently defined, and which return an error message, are:

- `union`
- `intersect`
- `setdiff`
- `punion`
- `pintersect`
- `psetdiff`

I am happy to add these methods if appropriate, so please contact me if you have suggestions for good definitions.

2.1.5 Additional methods unique to GTuples

GTuples have a few specifically defined methods that do not exist for *GenomicRanges::GRanges*. These are `tuples`, `size` and `IPD`.

The `tuples` method we have already seen and is somewhat analagous to the `ranges` method for *GenomicRanges::GRanges*, although returning an integer matrix rather than an *IRanges::IRanges* object:

```
tuples(gt3)
```

```
##       pos1 pos2 pos3
##  [1,]    1    2    3
##  [2,]    2    3    4
##  [3,]    3    4    5
##  [4,]    4    5    6
##  [5,]    5    6    7
##  [6,]    6    7    8
##  [7,]    7    8    9
##  [8,]    8    9   10
##  [9,]    9   10   11
## [10,]   10   11   12
```

The `size` method returns the size of the tuples stored in the object:

```
size(gt3)
```

```
## [1] 3
```

Every m-tuple with $m \geq 2$ has an associated vector of intra-pair distances (IPD). This is defined as $IPD = (pos_2 - pos_1, \ldots, pos_m - pos_{m-1}$. The `IPD` method returns this as an integer matrix, where the ith row contains the IPD for the ith tuple:

```
IPD(gt3)
```

```
##       [,1] [,2]
##  [1,]    1    1
##  [2,]    1    1
##  [3,]    1    1
```

```
##  [4,]    1    1
##  [5,]    1    1
##  [6,]    1    1
##  [7,]    1    1
##  [8,]    1    1
##  [9,]    1    1
## [10,]    1    1
```

2.2 Implementation details

While the *GTuple* class can be thought of as a matrix-link object, with the number of columns equal to the `size` of the tuples plus two (one for the `seqname` and one for the `strand`, internally, it extends the *GenomicRanges::GRanges* class. Specifically, the `ranges` slot stores an *IRanges::IRanges* object containing pos_1 and pos_m and, if `size` ¿ 2, a matrix is used to store the "internal" coordinates $pos_2, \ldots, pos_{m-1}$ in the `internalPos` slot. If `size`≤ 2 then the `internalPos` slot is set to `NULL`. The `size` is stored as an integer in the `size` slot.

While there are arguments for creating stand-alone *GTuples* and *GTuplesList* classes, by extending the *GRanges* and *GRangesList* classes I get a lot of very useful functionality "for free" via appropriately defined inheritance.

3 GTuplesList

The *GTuplesList* class is a container to store a *List* of *GTuples* objects. It extends the *GenomicRanges::GRangesList* class.

Currently, all *GTuples* in a *GTuplesList* must have the same `size`[1]. I expect that users will mostly use *GTuples* objects and have little need to directly use *GTuplesList* objects.

```
seqinfo <- Seqinfo(paste0("chr", 1:3), c(1000, 2000, 1500), NA, "mock1")
gt3 <- GTuples(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"),
                              c(1, 3, 2, 4)),
               tuples = matrix(c(1:10, 2:11, 3:12), ncol = 3),
               strand = Rle(strand(c("-", "+", "*", "+", "-")),
                            c(1, 2, 2, 3, 2)),
               score = 1:10, GC = seq(1, 0, length = 10), seqinfo = seqinfo)
gtl3 <- GTuplesList(A = gt3[1:5], B = gt3[6:10])
gtl3
```

```
## GTuplesList object of length 2:
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    1    2    3      - |     1                 1
##   [2]     chr2    2    3    4      + |     2 0.888888888888889
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    4    5    6      * |     4 0.666666666666667
##   [5]     chr1    5    6    7      * |     5 0.555555555555556
##
## $B
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    6    7    8      + |     6 0.444444444444444
```

[1]This may be changed in future versions of *GenomicTuples*.

```
##   [2]     chr3    7    8    9      + |     7 0.333333333333333
##   [3]     chr3    8    9   10      + |     8 0.222222222222222
##   [4]     chr3    9   10   11      - |     9 0.111111111111111
##   [5]     chr3   10   11   12      - |    10                 0
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

For even more information on the *GTuplesList* classes be sure to consult the manual page.

3.1 GTuplesList methods

Most methods defined for *GenomicRanges::GRangesList* are also applicable to *GTuplesList*. Those that are not yet defined, which are those that make sense for ranges but generally not for tuples, return error messages.

If a method that is not defined for *GTuplesList* but is defined for *GenomicRanges::GRangesList* is truly required, then this can be achieved by first coercing the *GTuplesList* object to a *GenomicRanges::GRangesList* object, noting that this is generally a destructive operation:

```
as(gtl3, "GRangesList")
```

```
## GRangesList object of length 2:
## $A
## GRanges object with 5 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]     chr1    [1, 3]      -
##   [2]     chr2    [2, 4]      +
##   [3]     chr2    [3, 5]      +
##   [4]     chr2    [4, 6]      *
##   [5]     chr1    [5, 7]      *
##
## $B
## GRanges object with 5 ranges and 0 metadata columns:
##       seqnames   ranges strand
##   [1]     chr1 [ 6,  8]      +
##   [2]     chr3 [ 7,  9]      +
##   [3]     chr3 [ 8, 10]      +
##   [4]     chr3 [ 9, 11]      -
##   [5]     chr3 [10, 12]      -
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

3.1.1 Basic GTuplesList accessors

These are very similar to those available for *GTuples* objects, except that they typically return a list since the input is now essentially a list of *GTuples* objects:

```
seqnames(gtl3)
```

```
## RleList of length 2
## $A
## factor-Rle of length 5 with 3 runs
```

```
##   Lengths:    1    3    1
##   Values : chr1 chr2 chr1
## Levels(3): chr1 chr2 chr3
##
## $B
## factor-Rle of length 5 with 2 runs
##   Lengths:    1    4
##   Values : chr1 chr3
## Levels(3): chr1 chr2 chr3
```

```
# Returns a list of integer matrices
tuples(gtl3)
```

```
## List of length 2
## names(2): A B
```

```
tuples(gtl3)[[1]]
```

```
##      pos1 pos2 pos3
## [1,]    1    2    3
## [2,]    2    3    4
## [3,]    3    4    5
## [4,]    4    5    6
## [5,]    5    6    7
```

```
strand(gtl3)
```

```
## RleList of length 2
## $A
## factor-Rle of length 5 with 3 runs
##   Lengths: 1 2 2
##   Values : - + *
## Levels(3): + - *
##
## $B
## factor-Rle of length 5 with 2 runs
##   Lengths: 3 2
##   Values : + -
## Levels(3): + - *
```

The `length` and `names` methods will return the length or names of the list:

```
length(gtl3)
```

```
## [1] 2
```

```
names(gtl3)
```

```
## [1] "A" "B"
```

Seqinfo can be extracted using the `seqinfo` accessor:

```
seqinfo(gtl3)
```

```
## Seqinfo object with 3 sequences from mock1 genome:
##   seqnames seqlengths isCircular genome
##   chr1           1000         NA  mock1
##   chr2           2000         NA  mock1
##   chr3           1500         NA  mock1
```

The `elementLengths` method returns a list of integers corresponding to the result of calling `length` on each individual

GTuples object contained by the *GTuplesList*. This is a faster alternative to calling `lapply` on the *GTuplesList*:

```
elementLengths(gtl3)

## A B
## 5 5
```

You can also use `isEmpty` to test if a *GTuplesList* object contains anything:

```
isEmpty(gtl3)

## [1] FALSE

isEmpty(GTuplesList())

## [1] TRUE
```

Finally, in the context of a *GTuplesList* object, the `mcols` method performs a similar operation to what it does on a *GTuples* object. However, this metadata now refers to information at the list level instead of the level of the individual *GTuples* objects:

```
mcols(gtl3) <- c("Feature A", "Feature B")
mcols(gtl3)

## DataFrame with 2 rows and 1 column
##         value
##   <character>
## 1   Feature A
## 2   Feature B
```

3.1.2 Combining GTuplesList objects

GTuplesList objects can be unlisted to combine the separate *GTuples* objects that they contain as an expanded *GTuples*:

```
ul <- unlist(gtl3)
ul

## GTuples object with 10 x 3-tuples and 2 metadata columns:
##     seqnames pos1 pos2 pos3 strand | score                GC
##   A     chr1    1    2    3      - |     1                 1
##   A     chr2    2    3    4      + |     2 0.888888888888889
##   A     chr2    3    4    5      + |     3 0.777777777777778
##   A     chr2    4    5    6      * |     4 0.666666666666667
##   A     chr1    5    6    7      * |     5 0.555555555555556
##   B     chr1    6    7    8      + |     6 0.444444444444444
##   B     chr3    7    8    9      + |     7 0.333333333333333
##   B     chr3    8    9   10      + |     8 0.222222222222222
##   B     chr3    9   10   11      - |     9 0.111111111111111
##   B     chr3   10   11   12      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

You can also append values together useing `append` or `c`.

3.1.3 Subsetting GTuplesList objects

Subsetting of *GTuplesList* objects is identical to subsetting of *GenomicRanges::GRangesList* objects:

```
gtl3[1]
```

```
## GTuplesList object of length 1:
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    1    2    3      - |     1                 1
##   [2]     chr2    2    3    4      + |     2 0.888888888888889
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    4    5    6      * |     4 0.666666666666667
##   [5]     chr1    5    6    7      * |     5 0.555555555555556
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

```
gtl3[[1]]
```

```
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    1    2    3      - |     1                 1
##   [2]     chr2    2    3    4      + |     2 0.888888888888889
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    4    5    6      * |     4 0.666666666666667
##   [5]     chr1    5    6    7      * |     5 0.555555555555556
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
gtl3["A"]
```

```
## GTuplesList object of length 1:
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    1    2    3      - |     1                 1
##   [2]     chr2    2    3    4      + |     2 0.888888888888889
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    4    5    6      * |     4 0.666666666666667
##   [5]     chr1    5    6    7      * |     5 0.555555555555556
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

```
gtl3$B
```

```
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    6    7    8      + |     6 0.444444444444444
##   [2]     chr3    7    8    9      + |     7 0.333333333333333
##   [3]     chr3    8    9   10      + |     8 0.222222222222222
##   [4]     chr3    9   10   11      - |     9 0.111111111111111
##   [5]     chr3   10   11   12      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

When subsetting a *GTuplesList*, you can also pass in a second parameter (as with a *GTuples* object) to again specify which of the metadata columns you wish to select:

```
gtl3[1, "score"]
```

```
## GTuplesList object of length 1:
## $A
## GTuples object with 5 x 3-tuples and 1 metadata column:
##       seqnames pos1 pos2 pos3 strand | score
##   [1]     chr1    1    2    3      - |     1
##   [2]     chr2    2    3    4      + |     2
##   [3]     chr2    3    4    5      + |     3
##   [4]     chr2    4    5    6      * |     4
##   [5]     chr1    5    6    7      * |     5
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

```
gtl3["B", "GC"]
```

```
## GTuplesList object of length 1:
## $B
## GTuples object with 5 x 3-tuples and 1 metadata column:
##       seqnames pos1 pos2 pos3 strand |                GC
##   [1]     chr1    6    7    8      + | 0.444444444444444
##   [2]     chr3    7    8    9      + | 0.333333333333333
##   [3]     chr3    8    9   10      + | 0.222222222222222
##   [4]     chr3    9   10   11      - | 0.111111111111111
##   [5]     chr3   10   11   12      - |                 0
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

The `head`, `tail`, `rep`, `rev`, and `window` methods all behave as you would expect them to for a list object. For example, the elements referred to by `window` are now list elements instead of *GTuples* elements:

```
rep(gtl3[[1]], times = 3)
```

```
## GTuples object with 15 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand   | score                GC
##    [1]     chr1    1    2    3      -   |     1                 1
##    [2]     chr2    2    3    4      +   |     2 0.888888888888889
##    [3]     chr2    3    4    5      +   |     3 0.777777777777778
##    [4]     chr2    4    5    6      *   |     4 0.666666666666667
##    [5]     chr1    5    6    7      *   |     5 0.555555555555556
##    ...      ...  ...  ...  ...    ... ...   ...               ...
##   [11]     chr1    1    2    3      -   |     1                 1
##   [12]     chr2    2    3    4      +   |     2 0.888888888888889
##   [13]     chr2    3    4    5      +   |     3 0.777777777777778
##   [14]     chr2    4    5    6      *   |     4 0.666666666666667
##   [15]     chr1    5    6    7      *   |     5 0.555555555555556
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
rev(gtl3)
```

```
## GTuplesList object of length 2:
## $B
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
```

```
##   [1]     chr1    6    7    8      + |     6 0.444444444444444
##   [2]     chr3    7    8    9      + |     7 0.333333333333333
##   [3]     chr3    8    9   10      + |     8 0.222222222222222
##   [4]     chr3    9   10   11      - |     9 0.111111111111111
##   [5]     chr3   10   11   12      - |    10                 0
##
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    1    2    3      - |     1                 1
##   [2]     chr2    2    3    4      + |     2 0.888888888888889
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    4    5    6      * |     4 0.666666666666667
##   [5]     chr1    5    6    7      * |     5 0.555555555555556
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

```r
head(gtl3, n = 1)
```

```
## GTuplesList object of length 1:
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    1    2    3      - |     1                 1
##   [2]     chr2    2    3    4      + |     2 0.888888888888889
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    4    5    6      * |     4 0.666666666666667
##   [5]     chr1    5    6    7      * |     5 0.555555555555556
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

```r
tail(gtl3, n = 1)
```

```
## GTuplesList object of length 1:
## $B
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    6    7    8      + |     6 0.444444444444444
##   [2]     chr3    7    8    9      + |     7 0.333333333333333
##   [3]     chr3    8    9   10      + |     8 0.222222222222222
##   [4]     chr3    9   10   11      - |     9 0.111111111111111
##   [5]     chr3   10   11   12      - |    10                 0
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

```r
window(gtl3, start = 1, end = 1)
```

```
## GTuplesList object of length 1:
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    1    2    3      - |     1                 1
##   [2]     chr2    2    3    4      + |     2 0.888888888888889
```

```
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    4    5    6      * |     4 0.666666666666667
##   [5]     chr1    5    6    7      * |     5 0.555555555555556
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

3.1.4 Basic tuple operations for GTuplesList objects

Basic tuple characteristics of *GTuplesList* objects can be extracted using the `start`, `end`, and `tuples` methods. These are very similar to those available for *GTuples* objects, except that they typically return a list since the input is now essentially a list of *GTuples* objects.

warning: While the `width` method is well-defined, namely it returns an `IntegerList` of $pos_m - pos_1 + 1$, this is not generally what is desired or required. Instead, please see the `IPD` method that will be discussed in the Section 3.1.6.

```
start(gtl3)

## IntegerList of length 2
## [["A"]] 1 2 3 4 5
## [["B"]] 6 7 8 9 10

end(gtl3)

## IntegerList of length 2
## [["A"]] 3 4 5 6 7
## [["B"]] 8 9 10 11 12

tuples(gtl3)

## List of length 2
## names(2): A B
```

Intra-tuple operations Most of the intra-range methods defined for *GenomicRanges::GRangesList* objects are not currently defined via extension for `GTuples` objects due to the differences between ranges and tuples. Those not currently defined, and which return an error message, are:

- `flank`
- `promoters`
- `resize`
- `restrict`

I am happy to add these methods if appropriate, so please contact me if you have suggestions for good definitions.

The `shift` method is well-defined:

```
shift(gtl3, 500)

## GTuplesList object of length 2:
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1  501  502  503      - |     1                 1
##   [2]     chr2  502  503  504      + |     2 0.888888888888889
##   [3]     chr2  503  504  505      + |     3 0.777777777777778
##   [4]     chr2  504  505  506      * |     4 0.666666666666667
##   [5]     chr1  505  506  507      * |     5 0.555555555555556
```

```
##
## $B
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1  506  507  508      + |     6 0.444444444444444
##   [2]     chr3  507  508  509      + |     7 0.333333333333333
##   [3]     chr3  508  509  510      + |     8 0.222222222222222
##   [4]     chr3  509  510  511      - |     9 0.111111111111111
##   [5]     chr3  510  511  512      - |    10                 0
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

```
shift(gtl3, IntegerList(A = 300L, B = 500L))
```

```
## GTuplesList object of length 2:
## $A
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1  301  302  303      - |     1                 1
##   [2]     chr2  302  303  304      + |     2 0.888888888888889
##   [3]     chr2  303  304  305      + |     3 0.777777777777778
##   [4]     chr2  304  305  306      * |     4 0.666666666666667
##   [5]     chr1  305  306  307      * |     5 0.555555555555556
##
## $B
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1  506  507  508      + |     6 0.444444444444444
##   [2]     chr3  507  508  509      + |     7 0.333333333333333
##   [3]     chr3  508  509  510      + |     8 0.222222222222222
##   [4]     chr3  509  510  511      - |     9 0.111111111111111
##   [5]     chr3  510  511  512      - |    10                 0
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

Inter-tuple operations None of the inter-range methods defined for *GenomicRanges::GRangesList* objects are currently defined via extension for *GTuplesList* objects due to the differences between ranges and tuples. Those not currently defined, and which return an error message, are:

- `range`
- `reduce`
- `disjoin`
- `isDisjoint`

I am happy to add these methods if appropriate, so please contact me if you have suggestions for good definitions.

Interval set operations for GTuplesList objects None of the interval set operations defined for *GenomicRanges::GRangesList* objects are currently defined via extension for *GTuplesList* objects due to the differences between ranges and tuples. Those not currently defined, and which return an error message, are:

- `punion`
- `pintersect`

- `psetdiff`

I am happy to add these methods if appropriate, so please contact me if you have suggestions for good definitions.

3.1.5 Looping over GTuplesList objects

Like for *GenomicRanges::GRangesList* objects, for *GTuplesList* objects there is a family of apply methods. These include `lapply`, `sapply`, `mapply`, `endoapply`, `mendoapply`, `Map`, and `Reduce`. The different looping methods defined for *GTuplesList* objects are useful for returning different kinds of results. The standard `lapply` and `sapply` behave according to convention, with the `lapply` method returning a list and `sapply` returning a more simplified output:

```
lapply(gtl3, length)

## $A
## [1] 5
##
## $B
## [1] 5

sapply(gtl3, length)

## A B
## 5 5
```

As with *GRangesList* objects, there is also a multivariate version of `sapply`, called `mapply`, defined for *GTuplesList* objects. And, if you don't want the results simplified, you can call the `Map` method, which does the same things as `mapply` but without simplifying the output:

```
gtl3_shift <- shift(gtl3, 10)
names(gtl3) <- c("shiftA", "shiftB")
mapply(c, gtl3, gtl3_shift)

## $shiftA
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand | score                GC
##    [1]     chr1    1    2    3      - |     1                 1
##    [2]     chr2    2    3    4      + |     2 0.888888888888889
##    [3]     chr2    3    4    5      + |     3 0.777777777777778
##    [4]     chr2    4    5    6      * |     4 0.666666666666667
##    [5]     chr1    5    6    7      * |     5 0.555555555555556
##    [6]     chr1   11   12   13      - |     1                 1
##    [7]     chr2   12   13   14      + |     2 0.888888888888889
##    [8]     chr2   13   14   15      + |     3 0.777777777777778
##    [9]     chr2   14   15   16      * |     4 0.666666666666667
##   [10]     chr1   15   16   17      * |     5 0.555555555555556
##   ---
##   seqinfo: 3 sequences from mock1 genome
##
## $shiftB
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand | score                GC
##    [1]     chr1    6    7    8      + |     6 0.444444444444444
##    [2]     chr3    7    8    9      + |     7 0.333333333333333
##    [3]     chr3    8    9   10      + |     8 0.222222222222222
##    [4]     chr3    9   10   11      - |     9 0.111111111111111
##    [5]     chr3   10   11   12      - |    10                 0
##    [6]     chr1   16   17   18      + |     6 0.444444444444444
```

```
##    [7]     chr3   17   18   19      + |     7 0.333333333333333
##    [8]     chr3   18   19   20      + |     8 0.222222222222222
##    [9]     chr3   19   20   21      - |     9 0.111111111111111
##   [10]     chr3   20   21   22      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

```
Map(c, gtl3, gtl3_shift)
```

```
## $shiftA
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand | score                GC
##    [1]     chr1    1    2    3      - |     1                 1
##    [2]     chr2    2    3    4      + |     2 0.888888888888889
##    [3]     chr2    3    4    5      + |     3 0.777777777777778
##    [4]     chr2    4    5    6      * |     4 0.666666666666667
##    [5]     chr1    5    6    7      * |     5 0.555555555555556
##    [6]     chr1   11   12   13      - |     1                 1
##    [7]     chr2   12   13   14      + |     2 0.888888888888889
##    [8]     chr2   13   14   15      + |     3 0.777777777777778
##    [9]     chr2   14   15   16      * |     4 0.666666666666667
##   [10]     chr1   15   16   17      * |     5 0.555555555555556
##   ---
##   seqinfo: 3 sequences from mock1 genome
##
## $shiftB
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand | score                GC
##    [1]     chr1    6    7    8      + |     6 0.444444444444444
##    [2]     chr3    7    8    9      + |     7 0.333333333333333
##    [3]     chr3    8    9   10      + |     8 0.222222222222222
##    [4]     chr3    9   10   11      - |     9 0.111111111111111
##    [5]     chr3   10   11   12      - |    10                 0
##    [6]     chr1   16   17   18      + |     6 0.444444444444444
##    [7]     chr3   17   18   19      + |     7 0.333333333333333
##    [8]     chr3   18   19   20      + |     8 0.222222222222222
##    [9]     chr3   19   20   21      - |     9 0.111111111111111
##   [10]     chr3   20   21   22      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

The `endoapply` method will return the results as a *GTuplesList* object rather than as a list:

```
endoapply(gtl3, rev)
```

```
## GTuplesList object of length 2:
## $shiftA
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr1    5    6    7      * |     5 0.555555555555556
##   [2]     chr2    4    5    6      * |     4 0.666666666666667
##   [3]     chr2    3    4    5      + |     3 0.777777777777778
##   [4]     chr2    2    3    4      + |     2 0.888888888888889
##   [5]     chr1    1    2    3      - |     1                 1
##
## $shiftB
```

```
## GTuples object with 5 x 3-tuples and 2 metadata columns:
##       seqnames pos1 pos2 pos3 strand | score                GC
##   [1]     chr3   10   11   12      - |    10                 0
##   [2]     chr3    9   10   11      - |     9 0.111111111111111
##   [3]     chr3    8    9   10      + |     8 0.222222222222222
##   [4]     chr3    7    8    9      + |     7 0.333333333333333
##   [5]     chr1    6    7    8      + |     6 0.444444444444444
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

There is also a multivariate version of the `endoapply` method in the form of the `mendoapply` method:

```
mendoapply(c, gtl3, gtl3_shift)
```

```
## GTuplesList object of length 2:
## $shiftA
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand | score                GC
##    [1]     chr1    1    2    3      - |     1                 1
##    [2]     chr2    2    3    4      + |     2 0.888888888888889
##    [3]     chr2    3    4    5      + |     3 0.777777777777778
##    [4]     chr2    4    5    6      * |     4 0.666666666666667
##    [5]     chr1    5    6    7      * |     5 0.555555555555556
##    [6]     chr1   11   12   13      - |     1                 1
##    [7]     chr2   12   13   14      + |     2 0.888888888888889
##    [8]     chr2   13   14   15      + |     3 0.777777777777778
##    [9]     chr2   14   15   16      * |     4 0.666666666666667
##   [10]     chr1   15   16   17      * |     5 0.555555555555556
##
## $shiftB
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand | score                GC
##    [1]     chr1    6    7    8      + |     6 0.444444444444444
##    [2]     chr3    7    8    9      + |     7 0.333333333333333
##    [3]     chr3    8    9   10      + |     8 0.222222222222222
##    [4]     chr3    9   10   11      - |     9 0.111111111111111
##    [5]     chr3   10   11   12      - |    10                 0
##    [6]     chr1   16   17   18      + |     6 0.444444444444444
##    [7]     chr3   17   18   19      + |     7 0.333333333333333
##    [8]     chr3   18   19   20      + |     8 0.222222222222222
##    [9]     chr3   19   20   21      - |     9 0.111111111111111
##   [10]     chr3   20   21   22      - |    10                 0
##
## -------
## seqinfo: 3 sequences from mock1 genome
```

Finally, the `Reduce` method will allow the *GTuples* objects to be collapsed across the whole of the *GTuplesList* object:

```
Reduce(c, gtl3)
```

```
## GTuples object with 10 x 3-tuples and 2 metadata columns:
##        seqnames pos1 pos2 pos3 strand | score                GC
##    [1]     chr1    1    2    3      - |     1                 1
##    [2]     chr2    2    3    4      + |     2 0.888888888888889
##    [3]     chr2    3    4    5      + |     3 0.777777777777778
```

```
##    [4]     chr2    4    5    6      * |     4 0.666666666666667
##    [5]     chr1    5    6    7      * |     5 0.555555555555556
##    [6]     chr1    6    7    8      + |     6 0.444444444444444
##    [7]     chr3    7    8    9      + |     7 0.333333333333333
##    [8]     chr3    8    9   10      + |     8 0.222222222222222
##    [9]     chr3    9   10   11      - |     9 0.111111111111111
##   [10]     chr3   10   11   12      - |    10                 0
##   ---
##   seqinfo: 3 sequences from mock1 genome
```

3.1.6 Additional methods unique to GTuplesList

Like *GTuples*, *GTuplesList* have a few specifically defined methods that do not exist for *GenomicRanges::GRangesList*. These are `tuples`, `size` and `IPD`. These are identical to the methods for *GTuples*, except that they typically return a list since the input is now essentially a list of *GTuples* objects.

```
tuples(gtl3)

## List of length 2
## names(2): shiftA shiftB

tuples(gtl3)[[1]]

##      pos1 pos2 pos3
## [1,]    1    2    3
## [2,]    2    3    4
## [3,]    3    4    5
## [4,]    4    5    6
## [5,]    5    6    7

size(gtl3)

## [1] 3

IPD(gtl3)

## List of length 2
## names(2): shiftA shiftB

IPD(gtl3)[[1]]

##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1
## [3,]    1    1
## [4,]    1    1
## [5,]    1    1
```

3.2 Implementation details

The *GTuplesList* class extends the *GenomicRanges::GRangesList* class.

4 findOverlaps-based methods

The definition of what constitutes an "overlap" between genomic tuples, or between genomic tuples and genomic ranges, lies at the heart of all findOverlaps-based methods[2] for *GTuples* and *GTuplesList* objects.

I have chosen a definition that matches my intuition of what constitutes an "overlap" between genomic tuples or between genomic tuples and genomic ranges. However, I am open to suggestions on amending or extending this behaviour in future versions of *GenomicTuples*.

4.1 Definition of overlapping genomic tuples

I consider two genomic tuples to be *equal* (type = "equal") if they have identical sequence names (seqnames), strands (strand) and tuples (tuples). For 1-tuples and 2-tuples, this means we can simply defer to the findOverlaps-based methods for *GenomicRanges::GRanges* and *GenomicRanges::GRangesList* objects via inheritance. However, we cannot do the same for m-tuples with $m > 2$ since this would ignore the "internal positions". Therefore, I have implemented a special case of the findOverlaps method for when size > 2 and type = "equal", which ensures that the "internal positions" are also checked for equality.

In all other cases genomic tuples are treated as genomic ranges. This means that when type = "any", type = "start", type = "end" or type = "within" then the genomic tuples are treated as if they were genomic ranges. Specifically, *GTuples* (resp. *GTuplesList*) are treated as though they were *GenomicRanges::GRanges* (resp. *GenomicRanges::GRangesList*) with pos1 = start and posm = end.

4.2 Definition of overlapping genomic tuples and ranges

Genomic tuples are **always** treated as genomic ranges when searching for overlaps between genomic tuples and genomic ranges.

4.3 Examples

It is easiest to understand the above definitions by studying a few examples.

Firstly, for 1-tuples where the *GTuples* methods use the *GRanges* methods:

```
# Construct example 1-tuples
gt1 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr2'),
               tuples = matrix(c(10L, 10L, 10L, 10L), ncol = 1),
               strand = c('+', '-', '*', '+'))
# GRanges version of gt1
gr1 <- as(gt1, "GRanges")
findOverlaps(gt1, gt1, type = 'any')
```

```
## Hits of length 8
## queryLength: 4
## subjectLength: 4
##   queryHits subjectHits
##    <integer>   <integer>
##  1         1           1
##  2         1           3
##  3         2           2
##  4         2           3
##  5         3           1
```

[2]The findOverlaps-based methods are findOverlaps, countOverlaps, overlapsAny and subsetByOverlaps.

```
##  6          3           2
##  7          3           3
##  8          4           4
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt1, gt1, type = 'any'),
          findOverlaps(gr1, gr1, type = 'any'))
```

```
## [1] TRUE
```

```
findOverlaps(gt1, gt1, type = 'start')
```

```
## Hits of length 8
## queryLength: 4
## subjectLength: 4
##   queryHits subjectHits
##    <integer>   <integer>
##  1          1           1
##  2          1           3
##  3          2           2
##  4          2           3
##  5          3           1
##  6          3           2
##  7          3           3
##  8          4           4
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt1, gt1, type = 'start'),
          findOverlaps(gr1, gr1, type = 'start'))
```

```
## [1] TRUE
```

```
findOverlaps(gt1, gt1, type = 'end')
```

```
## Hits of length 8
## queryLength: 4
## subjectLength: 4
##   queryHits subjectHits
##    <integer>   <integer>
##  1          1           1
##  2          1           3
##  3          2           2
##  4          2           3
##  5          3           1
##  6          3           2
##  7          3           3
##  8          4           4
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt1, gt1, type = 'end'),
          findOverlaps(gr1, gr1, type = 'end'))
```

```
## [1] TRUE
```

```
findOverlaps(gt1, gt1, type = 'within')
```

```
## Hits of length 8
## queryLength: 4
## subjectLength: 4
##   queryHits subjectHits
```

```
##    <integer>   <integer>
##  1         1           1
##  2         1           3
##  3         2           2
##  4         2           3
##  5         3           1
##  6         3           2
##  7         3           3
##  8         4           4
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt1, gt1, type = 'within'),
          findOverlaps(gr1, gr1, type = 'within'))
```

```
## [1] TRUE
```

```
findOverlaps(gt1, gt1, type = 'equal')
```

```
## Hits of length 8
## queryLength: 4
## subjectLength: 4
##   queryHits subjectHits
##    <integer>   <integer>
##  1         1           1
##  2         1           3
##  3         2           2
##  4         2           3
##  5         3           1
##  6         3           2
##  7         3           3
##  8         4           4
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt1, gt1, type = 'equal'),
          findOverlaps(gr1, gr1, type = 'equal'))
```

```
## [1] TRUE
```

```
# Can pass other arguments, such as select and ignore.strand
findOverlaps(gt1, gt1, type = 'equal', ignore.strand = TRUE, select = 'last')
```

```
## [1] 3 3 3 4
```

Next, for 2-tuples where the *GTuples* methods use the *GRanges* methods:

```
# Construct example 2-tuples
gt2 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr1', 'chr2'),
               tuples = matrix(c(10L, 10L, 10L, 10L, 10L, 20L, 20L, 20L, 25L,
                                 20L), ncol = 2),
               strand = c('+', '-', '*', '+', '+'))
# GRanges version of gt2
gr2 <- as(gt2, "GRanges")
findOverlaps(gt2, gt2, type = 'any')
```

```
## Hits of length 13
## queryLength: 5
## subjectLength: 5
##     queryHits subjectHits
##      <integer>   <integer>
```

```
##  1           1           1
##  2           1           3
##  3           1           4
##  4           2           2
##  5           2           3
##  ...       ...         ...
##  9           3           4
##  10          4           1
##  11          4           3
##  12          4           4
##  13          5           5
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt2, gt2, type = 'any'),
          findOverlaps(gr2, gr2, type = 'any'))
```

```
## [1] TRUE
```

```
findOverlaps(gt2, gt2, type = 'start')
```

```
## Hits of length 13
## queryLength: 5
## subjectLength: 5
##     queryHits subjectHits
##      <integer>   <integer>
##  1           1           1
##  2           1           3
##  3           1           4
##  4           2           2
##  5           2           3
##  ...       ...         ...
##  9           3           4
##  10          4           1
##  11          4           3
##  12          4           4
##  13          5           5
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt2, gt2, type = 'start'),
          findOverlaps(gr2, gr2, type = 'start'))
```

```
## [1] TRUE
```

```
findOverlaps(gt2, gt2, type = 'end')
```

```
## Hits of length 9
## queryLength: 5
## subjectLength: 5
##   queryHits subjectHits
##    <integer>   <integer>
##  1         1           1
##  2         1           3
##  3         2           2
##  4         2           3
##  5         3           1
##  6         3           2
##  7         3           3
##  8         4           4
```

```
##  9         5           5

# GTuples and GRanges methods identical
identical(findOverlaps(gt2, gt2, type = 'end'),
          findOverlaps(gr2, gr2, type = 'end'))

## [1] TRUE

findOverlaps(gt2, gt2, type = 'within')

## Hits of length 11
## queryLength: 5
## subjectLength: 5
##     queryHits subjectHits
##      <integer>   <integer>
##  1           1           1
##  2           1           3
##  3           1           4
##  4           2           2
##  5           2           3
##  ...       ...         ...
##  7           3           2
##  8           3           3
##  9           3           4
##  10          4           4
##  11          5           5

# GTuples and GRanges methods identical
identical(findOverlaps(gt2, gt2, type = 'within'),
          findOverlaps(gr2, gr2, type = 'within'))

## [1] TRUE

findOverlaps(gt2, gt2, type = 'equal')

## Hits of length 9
## queryLength: 5
## subjectLength: 5
##   queryHits subjectHits
##    <integer>   <integer>
##  1         1           1
##  2         1           3
##  3         2           2
##  4         2           3
##  5         3           1
##  6         3           2
##  7         3           3
##  8         4           4
##  9         5           5

# GTuples and GRanges methods identical
identical(findOverlaps(gt2, gt2, type = 'equal'),
          findOverlaps(gr2, gr2, type = 'equal'))

## [1] TRUE

# Can pass other arguments, such as select and ignore.strand
findOverlaps(gt2, gt2, type = 'equal', ignore.strand = TRUE, select = 'last')

## [1] 3 3 3 4 5
```

Finally, for m-tuples with $m > 2$ where *GTuples* methods use the *GRanges* methods **unless** `size = "equal"`:

```
# Construct example 3-tuples
gt3 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr1', 'chr2'),
               tuples = matrix(c(10L, 10L, 10L, 10L, 10L, 20L, 20L, 20L, 25L,
                                 20L, 30L, 30L, 35L, 30L, 30L), ncol = 3),
               strand = c('+', '-', '*', '+', '+'))
# GRanges version of gt3
gr3 <- as(gt3, "GRanges")
findOverlaps(gt3, gt3, type = 'any')
```

```
## Hits of length 13
## queryLength: 5
## subjectLength: 5
##     queryHits subjectHits
##      <integer>   <integer>
##  1           1           1
##  2           1           3
##  3           1           4
##  4           2           2
##  5           2           3
##  ...       ...         ...
##  9           3           4
##  10          4           1
##  11          4           3
##  12          4           4
##  13          5           5
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt3, gt3, type = 'any'),
          findOverlaps(gr3, gr3, type = 'any')) # TRUE
```

```
## [1] TRUE
```

```
findOverlaps(gt3, gt3, type = 'start')
```

```
## Hits of length 13
## queryLength: 5
## subjectLength: 5
##     queryHits subjectHits
##      <integer>   <integer>
##  1           1           1
##  2           1           3
##  3           1           4
##  4           2           2
##  5           2           3
##  ...       ...         ...
##  9           3           4
##  10          4           1
##  11          4           3
##  12          4           4
##  13          5           5
```

```
# GTuples and GRanges methods identical
identical(findOverlaps(gt3, gt3, type = 'start'),
          findOverlaps(gr3, gr3, type = 'start')) # TRUE
```

```
## [1] TRUE
```

```r
findOverlaps(gt3, gt3, type = 'end')
```

```
## Hits of length 7
## queryLength: 5
## subjectLength: 5
##   queryHits subjectHits
##    <integer>   <integer>
##  1         1           1
##  2         1           4
##  3         2           2
##  4         3           3
##  5         4           1
##  6         4           4
##  7         5           5
```

```r
# GTuples and GRanges methods identical
identical(findOverlaps(gt3, gt3, type = 'end'),
          findOverlaps(gr3, gr3, type = 'end')) # TRUE
```

```
## [1] TRUE
```

```r
findOverlaps(gt3, gt3, type = 'within')
```

```
## Hits of length 10
## queryLength: 5
## subjectLength: 5
##    queryHits subjectHits
##     <integer>   <integer>
##  1          1           1
##  2          1           3
##  3          1           4
##  4          2           2
##  5          2           3
##  6          3           3
##  7          4           1
##  8          4           3
##  9          4           4
##  10         5           5
```

```r
# GTuples and GRanges methods identical
identical(findOverlaps(gt3, gt3, type = 'within'),
          findOverlaps(gr3, gr3, type = 'within')) # TRUE
```

```
## [1] TRUE
```

```r
findOverlaps(gt3, gt3, type = 'equal')
```

```
## Hits of length 5
## queryLength: 5
## subjectLength: 5
##   queryHits subjectHits
##    <integer>   <integer>
##  1         1           1
##  2         2           2
##  3         3           3
##  4         4           4
```

```
##  5         5           5

# GTuples and GRanges methods **not** identical because  GRanges method ignores
# "internal positions".
identical(findOverlaps(gt3, gt3, type = 'equal'),
          findOverlaps(gr3, gr3, type = 'equal')) # FALSE

## [1] FALSE

# Can pass other arguments, such as select and ignore.strand
findOverlaps(gt3, gt3, type = 'equal', ignore.strand = TRUE, select = 'last')

## [1] 2 2 3 4 5
```

5 Comparison of genomic tuples

I have chosen a definition that matches my intuition of what constitutes a comparison between genomic tuples. However, I am open to suggestions on amending or extending this behaviour in future versions of *GenomicTuples*.

5.1 Definition of comparison methods for tuples

The comparison of two genomic tuples, `x` and `y`, is done by first comparing the `seqnames(x)` to `seqnames(x)`, then `strand(x)` to `strand(x)` and finally `tuples(x)` to `tuples(x)`.

Ordering of `seqnames` and `strand` is as implemented *GenomicRanges::GRanges*. Ordering of `tuples` is element-wise, i.e. $pos_1, \ldots, pos_m$ are compared in turn. For example, `chr1:+:{10, 20, 30}` is considered less than `chr1:+:{10, 20, 40}`. This defines what I will refer to as the "natural order" of genomic tuples.

The above is implemented in the `compare` method for *GTuples*, which performs "generalized range-wise comparison" of two *GTuples* objects, `x` and `y`. That is, `compare(x, y)` returns an integer vector where the ith element is a code describing how the ith element in `x` is qualitatively positioned relatively to the ith element in `y`. A code that is `< 0`, `= 0`, or `> 0`, corresponds to `x[i] < y[i]`, `x[i] == y[i]`, or `x[i] > y[i]`, respectively.

The 6 traditional binary comparison operators (`==`, `!=`, `<=`, `>=`, `<`, and `>`), other comparison operators (`match`, `order`, `sort` and `rank`) and duplicate-based methods (`duplicated` and `unique`) all use this "natural order".

5.2 Examples

It is easiest to understand the above definitions by studying a few examples, here using 3-tuples:

```
# Construct example 3-tuples
gt3 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr1', 'chr2', 'chr1',
                            'chr1'),
               tuples = matrix(c(10L, 10L, 10L, 10L, 10L, 5L, 10L, 20L, 20L,
                                 20L, 25L, 20L, 20L, 20L, 30L, 30L, 35L, 30L,
                                 30L, 30L, 35L),
                               ncol = 3),
               strand = c('+', '-', '*', '+', '+', '+', '+'))
gt3

## GTuples object with 7 x 3-tuples and 0 metadata columns:
##       seqnames pos1 pos2 pos3 strand
##   [1]     chr1   10   20   30      +
##   [2]     chr1   10   20   30      -
```

```
##   [3]     chr1   10   20   35      *
##   [4]     chr1   10   25   30      +
##   [5]     chr2   10   20   30      +
##   [6]     chr1    5   20   30      +
##   [7]     chr1   10   20   35      +
##   ---
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
# Compare each tuple to itself
compare(gt3, gt3)
```

```
## [1] 0 0 0 0 0 0 0
```

```
gt3 < gt3
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
gt3 > gt3
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
gt3 == gt3
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
# Compare the third tuple to all tuples
compare(gt3[3], gt3)
```

```
## [1]  2  1  0  2 -1  2  2
```

```
gt3[3] < gt3
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

```
gt3[3] > gt3
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
```

```
gt3[3] == gt3
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

```
## Some comparisons where tuples differ only in one coordinate

# Ordering of seqnames
# 'chr1' < 'chr2' for tuples with otherwise identical coordinates
gt3[1] < gt3[5] # TRUE
```

```
## [1] TRUE
```

```
# Ordering of strands
# '+' < '-' < '*' for tuples with otherwise identical coordiantes
gt3[1] < gt3[2] # TRUE
```

```
## [1] TRUE
```

```
gt3[1] < gt3[2] # TRUE
```

```
## [1] TRUE
```

```
gt3[1] < unstrand(gt3[2]) # TRUE
```

```
## [1] TRUE
```

```
gt3[2] < unstrand(gt3[2]) # TRUE
```

```
## [1] TRUE

# Ordering of tuples
# Tuples checked sequentially from pos1, ..., posm for tuples with otherwise
# identical coordinates
gt3[6] < gt3[1] # TRUE due to pos1

## [1] TRUE

gt3[2] < gt3[4] # TRUE due to pos2

## [1] FALSE

gt3[1] < gt3[7] # TRUE due to pos3

## [1] TRUE

# Sorting of tuples
# Sorted first by seqnames, then by strand, then by tuples
sort(gt3)

## GTuples object with 7 x 3-tuples and 0 metadata columns:
##       seqnames pos1 pos2 pos3 strand
##   [1]     chr1    5   20   30      +
##   [2]     chr1   10   20   30      +
##   [3]     chr1   10   20   35      +
##   [4]     chr1   10   25   30      +
##   [5]     chr1   10   20   30      -
##   [6]     chr1   10   20   35      *
##   [7]     chr2   10   20   30      +
##   ---
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths

# Duplicate tuples
# Duplicate tuples must have identical seqnames, strand and positions (tuples)
duplicated(c(gt3, gt3[1:3]))

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE

unique(c(gt3, gt3[1:3]))

## GTuples object with 7 x 3-tuples and 0 metadata columns:
##       seqnames pos1 pos2 pos3 strand
##   [1]     chr1   10   20   30      +
##   [2]     chr1   10   20   30      -
##   [3]     chr1   10   20   35      *
##   [4]     chr1   10   25   30      +
##   [5]     chr2   10   20   30      +
##   [6]     chr1    5   20   30      +
##   [7]     chr1   10   20   35      +
##   ---
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

6 Acknowledgements

I am very grateful to all the *Bioconductor* developers but particularly wish to thank the developers of *GenomicRanges* [1], which *GenomicTuples* uses heavily and is based upon.

7 Session info

Here is the output of `sessionInfo` on the system on which this document was compiled:

```
toLatex(sessionInfo())
```

- R version 3.1.1 Patched (2014-09-25 r66681), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.12.0, GenomeInfoDb 1.2.0, GenomicRanges 1.18.0, GenomicTuples 1.0.0, IRanges 2.0.0, S4Vectors 0.4.0
- Loaded via a namespace (and not attached): Biobase 2.26.0, BiocStyle 1.4.0, Rcpp 0.11.3, XVector 0.6.0, evaluate 0.5.5, formatR 1.0, highr 0.3, knitr 1.7, stringr 0.6.2, tools 3.1.1

References

[1] Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin Morgan, and Vincent Carey. Software for computing and annotating genomic ranges. *PLoS Computational Biology*, 9, 2013. URL: http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.pcbi.1003118, doi:10.1371/journal.pcbi.1003118.