

# Package: superintervalsr (via r-universe)

October 14, 2026

**Type** Package

**Title** Fast, Memory-Efficient Interval Operations for Genomics

**Version** 0.99.7

**Description** Provides a fast, memory-efficient data structure for interval intersection queries using a novel superset-index approach. SuperIntervals maintains intervals in position-sorted order, enabling cache-friendly searches and SIMD-optimized counting operations. Chromosome- and strand-aware indexes support repeated genomic queries while retaining GRanges metadata.

**License** MIT + file LICENSE

**URL** <https://github.com/kcleal/superintervalsr>

**BugReports** <https://github.com/kcleal/superintervalsr/issues>

**biocViews** Infrastructure, DataRepresentation, GenomeAnnotation, Sequencing

**Depends** R (>= 4.5.0)

**Imports** Rcpp (>= 1.0.0), S4Vectors, IRanges, GenomicRanges, methods

**LinkingTo** Rcpp

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0), airway, SummarizedExperiment

**SystemRequirements** C++17

**Encoding** UTF-8

**Language** en-US

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.1.1

**Config/testthat/edition** 3

**Roxygen** list(markdown = TRUE)

**Repository** <https://biocstaging.r-universe.dev>

**Date/Publication** 2026-10-14 11:58:48 UTC

**RemoteUrl** <https://github.com/BiocStaging/superintervalsr>

**RemoteRef** HEAD

**RemoteSha** cdc952b28e44282145030dc9058d33fd7309bbf6

## Contents

|                                                        |    |
|--------------------------------------------------------|----|
| [.IntervalMap . . . . .                                | 3  |
| add . . . . .                                          | 3  |
| as_granges . . . . .                                   | 4  |
| at . . . . .                                           | 4  |
| build . . . . .                                        | 5  |
| clear . . . . .                                        | 6  |
| count . . . . .                                        | 6  |
| count.batch . . . . .                                  | 7  |
| coverage . . . . .                                     | 8  |
| data_at . . . . .                                      | 8  |
| difference . . . . .                                   | 9  |
| ends_at . . . . .                                      | 10 |
| expand . . . . .                                       | 10 |
| findOverlaps,GRanges,SuperIntervalMap-method . . . . . | 11 |
| flank . . . . .                                        | 12 |
| gaps . . . . .                                         | 13 |
| has_overlaps . . . . .                                 | 13 |
| intersection . . . . .                                 | 14 |
| IntervalMap . . . . .                                  | 15 |
| length.IntervalMap . . . . .                           | 16 |
| merge_overlaps . . . . .                               | 17 |
| print.IntervalMap . . . . .                            | 17 |
| reserve . . . . .                                      | 18 |
| search_idxxs . . . . .                                 | 18 |
| search_idxxs.batch . . . . .                           | 19 |
| search_items . . . . .                                 | 20 |
| search_keys . . . . .                                  | 20 |
| search_point . . . . .                                 | 21 |
| search_values . . . . .                                | 22 |
| size . . . . .                                         | 22 |
| span . . . . .                                         | 23 |
| starts_at . . . . .                                    | 24 |
| SuperIntervalMap . . . . .                             | 24 |
| SuperIntervalMap-class . . . . .                       | 25 |
| symmetric_difference . . . . .                         | 26 |
| union_with . . . . .                                   | 27 |
| unique_intervals . . . . .                             | 27 |

---

|               |                                          |
|---------------|------------------------------------------|
| [.IntervalMap | <i>Access interval at specific index</i> |
|---------------|------------------------------------------|

---

**Description**

Access interval at specific index

**Usage**

```
## S3 method for class 'IntervalMap'  
x[i]
```

**Arguments**

|   |                      |
|---|----------------------|
| x | IntervalMap object   |
| i | Index (1-based in R) |

**Value**

List with start, end, and value components

---

|     |                                   |
|-----|-----------------------------------|
| add | <i>Add an interval to the map</i> |
|-----|-----------------------------------|

---

**Description**

Add an interval to the map

**Usage**

```
add(x, start, end, value = NULL)
```

**Arguments**

|       |                             |
|-------|-----------------------------|
| x     | IntervalMap object          |
| start | Start position (inclusive)  |
| end   | End position (inclusive)    |
| value | Associated value (optional) |

**Value**

The IntervalMap object (invisibly)

**Examples**

```
im <- IntervalMap()  
add(im, 1, 10, "gene1")  
add(im, 15, 25, "gene2")
```

---

|            |                                                         |
|------------|---------------------------------------------------------|
| as_granges | <i>Convert IntervalMap items result back to GRanges</i> |
|------------|---------------------------------------------------------|

---

**Description**

Helper function to convert search\_items results back to Bioconductor objects.

**Usage**

```
as_granges(items_result, seqname = "chr1", valuename = "value")
```

**Arguments**

|              |                                                                                                                               |
|--------------|-------------------------------------------------------------------------------------------------------------------------------|
| items_result | Result from search_items(), or a SuperIntervalMap. For an indexed genomic map, return its original GRanges with all metadata. |
| seqname      | Chromosome name to assign (default: "chr1")                                                                                   |
| valuename    | Value name to assign (default: "value")                                                                                       |

**Value**

GRanges object

**Examples**

```
# Create an IntervalMap and search for items
im <- IntervalMap(c(1, 10), ends = c(5, 15), values = c("gene1", "gene2"))
items <- search_items(im, 3, 12)

# Convert result back to GRanges
gr <- as_granges(items, seqname = "chr1", valuename = "gene_id")
print(gr)
```

---

|    |                                       |
|----|---------------------------------------|
| at | <i>Get interval at specific index</i> |
|----|---------------------------------------|

---

**Description**

Get interval at specific index

**Usage**

```
at(x, idx)
```

**Arguments**

|     |                          |
|-----|--------------------------|
| x   | IntervalMap object       |
| idx | Index position (1-based) |

**Value**

List with start, end, and value components

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
build(im)
at(im, 1)
```

---

build

*Build index for efficient queries*

---

**Description**

Must be called after adding intervals and before performing queries. Sorts by start ascending and end descending within tied starts. Accessor and search indices refer to this order.

**Usage**

```
build(x)
```

**Arguments**

x                      IntervalMap object

**Value**

The IntervalMap object (invisibly)

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
build(im)
```

---

|       |                            |
|-------|----------------------------|
| clear | <i>Clear all intervals</i> |
|-------|----------------------------|

---

**Description**

Clear all intervals

**Usage**

```
clear(x)
```

**Arguments**

|   |                    |
|---|--------------------|
| x | IntervalMap object |
|---|--------------------|

**Value**

The IntervalMap object (invisibly)

The IntervalMap object (invisibly)

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
clear(im)
```

---

|       |                                    |
|-------|------------------------------------|
| count | <i>Count overlapping intervals</i> |
|-------|------------------------------------|

---

**Description**

Count overlapping intervals

**Usage**

```
count(x, start, end)
```

**Arguments**

|       |                                  |
|-------|----------------------------------|
| x     | IntervalMap object               |
| start | Start of query range (inclusive) |
| end   | End of query range (inclusive)   |

**Value**

Number of overlapping intervals

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
build(im)
count(im, 7, 12) # 2
```

---

`count.batch`*Count overlaps for multiple ranges*

---

**Description**

Count overlaps for multiple ranges

**Usage**

```
count.batch(x, starts, ends)
```

**Arguments**

|                     |                                 |
|---------------------|---------------------------------|
| <code>x</code>      | IntervalMap object              |
| <code>starts</code> | Vector of query start positions |
| <code>ends</code>   | Vector of query end positions   |

**Value**

Integer vector of overlap counts, one per query

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
add(im, 20, 30, "gene3")
build(im)
count.batch(im, c(7, 25), c(12, 35)) # c(2, 1)
```

---

|          |                                            |
|----------|--------------------------------------------|
| coverage | <i>Get coverage statistics for a range</i> |
|----------|--------------------------------------------|

---

**Description**

Get coverage statistics for a range

**Usage**

coverage(x, start, end)

**Arguments**

- x                   IntervalMap object
- start               Start of query range (inclusive)
- end                 End of query range (inclusive)

**Details**

total\_coverage sums the inclusive widths of each intersection with the query. Overlapping stored intervals are counted independently; this is not the width of their union or an IRanges coverage Rle.

**Value**

List with integer count and numeric total\_coverage

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
build(im)
coverage(im, 1, 20) # Returns list with coverage statistics
```

---

|         |                                   |
|---------|-----------------------------------|
| data_at | <i>Get data at specific index</i> |
|---------|-----------------------------------|

---

**Description**

Get data at specific index

**Usage**

data\_at(x, idx)

**Arguments**

|     |                          |
|-----|--------------------------|
| x   | IntervalMap object       |
| idx | Index position (1-based) |

**Value**

Associated data value  
Associated data value at the specified index

**Examples**

```
im <- IntervalMap()  
add(im, 1, 10, "gene1")  
build(im)  
data_at(im, 1) # Returns "gene1"
```

---

|            |                                        |
|------------|----------------------------------------|
| difference | <i>Difference of two interval sets</i> |
|------------|----------------------------------------|

---

**Description**

Returns a new IntervalMap of regions in x that are not covered by y.

**Usage**

```
difference(x, y)
```

**Arguments**

|   |                        |
|---|------------------------|
| x | IntervalMap object (A) |
| y | IntervalMap object (B) |

**Value**

A new IntervalMap representing  $A \setminus B$

**Examples**

```
a <- IntervalMap(c(1, 20), ends = c(10, 30), values = c("A", "B"))  
b <- IntervalMap(c(5, 25), ends = c(15, 35), values = c("C", "D"))  
d <- difference(a, b)
```

---

|         |                                           |
|---------|-------------------------------------------|
| ends_at | <i>Get end position at specific index</i> |
|---------|-------------------------------------------|

---

**Description**

Get end position at specific index

**Usage**

```
ends_at(x, idx)
```

**Arguments**

|     |                          |
|-----|--------------------------|
| x   | IntervalMap object       |
| idx | Index position (1-based) |

**Value**

End position  
Integer end position of the interval

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
build(im)
ends_at(im, 1) # Returns 10
```

---

|        |                                              |
|--------|----------------------------------------------|
| expand | <i>Expand each interval by fixed amounts</i> |
|--------|----------------------------------------------|

---

**Description**

Grows (or shrinks) every interval by a fixed amount. Results are clamped to [lo, hi]. Intervals shrunk past themselves are dropped.

**Usage**

```
expand(x, left, right, lo = 0L, hi = 2147483647L)
```

**Arguments**

|       |                                                                            |
|-------|----------------------------------------------------------------------------|
| x     | IntervalMap object                                                         |
| left  | Amount to subtract from each start (extends to the left). May be negative. |
| right | Amount to add to each end (extends to the right). May be negative.         |
| lo    | Lower clamp (default is 0)                                                 |
| hi    | Upper clamp (default is the maximum 32-bit integer)                        |

**Value**

A new IntervalMap with expanded intervals

**Examples**

```
im <- IntervalMap(c(10, 50), ends = c(20, 60), values = c("A", "B"))
expanded <- expand(im, 5, 5)
```

---

findOverlaps,GRanges,SuperIntervalMap-method  
*Query an indexed genomic subject*

---

**Description**

Query an indexed genomic subject

**Usage**

```
## S4 method for signature 'GRanges,SuperIntervalMap'
findOverlaps(
  query,
  subject,
  maxgap = -1L,
  minoverlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  select = c("all", "first", "last", "arbitrary"),
  ignore.strand = FALSE
)

## S4 method for signature 'GRanges,SuperIntervalMap'
countOverlaps(
  query,
  subject,
  maxgap = -1L,
  minoverlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore.strand = FALSE
)
```

**Arguments**

|                                  |                                                                        |
|----------------------------------|------------------------------------------------------------------------|
| query                            | A GRanges of query features.                                           |
| subject                          | A SuperIntervalMap constructed once from the subject GRanges.          |
| maxgap, minoverlap, type, select | Overlap options, as in <a href="#">GenomicRanges::findOverlaps()</a> . |
| ignore.strand                    | Whether to ignore strand (default FALSE).                              |

**Value**

`findOverlaps()` returns a Hits object with indices in the original input order, or a subject-index vector when `select` is not "all". `countOverlaps()` returns one integer count per query feature.

**Examples**

```
library(GenomicRanges)
subject <- GRanges(c("chr1", "chr2"), IRanges(c(1, 10), c(5, 20)))
query <- GRanges(c("chr1", "chr2"), IRanges(c(3, 15), c(8, 18)))
index <- SuperIntervalMap(subject)
hits <- findOverlaps(query, index)
as_granges(index)[subjectHits(hits)]
countOverlaps(query, index)
```

---

flank

---

*Create flanking intervals beside each interval*


---

**Description**

Returns a new IntervalMap containing only the flanking intervals, not the originals. Left flanks are `[start - left, start - 1]` and right flanks are `[end + 1, end + right]`. Zero-width or out-of-range flanks are skipped.

**Usage**

```
flank(x, left, right, lo = 0L, hi = 2147483647L)
```

**Arguments**

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| <code>x</code>     | IntervalMap object                                  |
| <code>left</code>  | Width of the left flank (0 = none)                  |
| <code>right</code> | Width of the right flank (0 = none)                 |
| <code>lo</code>    | Lower clamp (default is 0)                          |
| <code>hi</code>    | Upper clamp (default is the maximum 32-bit integer) |

**Value**

A new IntervalMap containing the flank intervals

**Examples**

```
im <- IntervalMap(c(10, 50), ends = c(20, 60), values = c("A", "B"))
f <- flank(im, 5, 5)
```

---

|      |                                 |
|------|---------------------------------|
| gaps | <i>Find gaps within a range</i> |
|------|---------------------------------|

---

**Description**

Returns uncovered regions within the supplied range as a new IntervalMap. Gap intervals are assigned the value given by fill (default NULL).

**Usage**

```
gaps(x, lo, hi, fill = NULL)
```

**Arguments**

|      |                                                |
|------|------------------------------------------------|
| x    | IntervalMap object                             |
| lo   | Lower bound of the span to examine (inclusive) |
| hi   | Upper bound of the span to examine (inclusive) |
| fill | Value to assign to gap intervals               |

**Value**

A new IntervalMap of gap intervals

**Examples**

```
im <- IntervalMap(c(1, 10), ends = c(5, 15), values = c("A", "B"))
g <- gaps(im, 0, 20)
```

---

|              |                                                    |
|--------------|----------------------------------------------------|
| has_overlaps | <i>Check if any intervals overlap with a range</i> |
|--------------|----------------------------------------------------|

---

**Description**

Check if any intervals overlap with a range

**Usage**

```
has_overlaps(x, start, end)
```

**Arguments**

|       |                                  |
|-------|----------------------------------|
| x     | IntervalMap object               |
| start | Start of query range (inclusive) |
| end   | End of query range (inclusive)   |

**Value**

TRUE if any intervals overlap, FALSE otherwise

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
build(im)
has_overlaps(im, 5, 15) # TRUE
```

---

|              |                                          |
|--------------|------------------------------------------|
| intersection | <i>Intersection of two interval sets</i> |
|--------------|------------------------------------------|

---

**Description**

Returns a new IntervalMap of regions covered by both inputs. Output pieces are not coalesced; call `merge_overlaps()` on the result if a disjoint set is required. Each piece keeps the value from `x`.

**Usage**

```
intersection(x, y)
```

**Arguments**

|                |                    |
|----------------|--------------------|
| <code>x</code> | IntervalMap object |
| <code>y</code> | IntervalMap object |

**Value**

A new IntervalMap representing the intersection

**Examples**

```
a <- IntervalMap(c(1, 20), ends = c(10, 30), values = c("A", "B"))
b <- IntervalMap(c(5, 25), ends = c(15, 35), values = c("C", "D"))
i <- intersection(a, b)
```

---

|             |                                 |
|-------------|---------------------------------|
| IntervalMap | <i>Create a new IntervalMap</i> |
|-------------|---------------------------------|

---

## Description

Creates a new IntervalMap object for storing and querying intervals efficiently. This function uses method dispatch to automatically handle different input types.

## Usage

```
IntervalMap(x, ...)

## Default S3 method:
IntervalMap(x = NULL, ...)

## S3 method for class 'integer'
IntervalMap(x, ends, values = NULL, ...)

## S3 method for class 'numeric'
IntervalMap(x, ends, values = NULL, ...)

## S3 method for class 'GRanges'
IntervalMap(x, value_column = NULL, seqname = NULL, ...)

## S3 method for class 'data.frame'
IntervalMap(
  x,
  start_column = "start",
  end_column = "end",
  value_column = NULL,
  ...
)
```

## Arguments

|              |                                                                                                    |
|--------------|----------------------------------------------------------------------------------------------------|
| x            | Input object. Can be missing (creates empty map), GRanges, data.frame, or integer vector of starts |
| ...          | Additional arguments passed to specific methods                                                    |
| ends         | Integer vector of end positions (when x is numeric vector of starts)                               |
| values       | Optional list of values to associate with each interval                                            |
| value_column | Name of column to use as values (default: use all other columns as named list)                     |
| seqname      | Optional chromosome name to filter to (e.g., "chr1")                                               |
| start_column | Name of start column (default: "start")                                                            |
| end_column   | Name of end column (default: "end")                                                                |

## Details

Coordinates are closed (both endpoints inclusive) whole numbers in the R integer range, excluding NA. A map stores one coordinate space and ignores strand. Multi-sequence GRanges require an explicit seqname filter. Constructors build the index automatically. After add(), call build() before querying; queries do not check index state or validate coordinates. Query inputs must be valid whole-number ranges. Build sorts by start ascending, then end descending; indices refer to this internal order, not the original input row order. Query result order is unspecified. Store original row IDs as values when mapping hits back to input data. Mutation operates by reference, so assignment does not copy a map. External pointers cannot be restored with saveRDS/readRDS; save coordinates and values and recreate the map instead.

## Value

An IntervalMap object

## Examples

```
# Create empty IntervalMap
im1 <- IntervalMap()

# Create from GRanges
library(GenomicRanges)
gr <- GRanges("chr1", IRanges(c(1, 10), c(5, 15)), gene_id = c("A", "B"))
im2 <- IntervalMap(gr, value_column = "gene_id")

# Create from data.frame
df <- data.frame(start = c(1, 10), end = c(5, 15), gene = c("A", "B"))
im3 <- IntervalMap(df, value_column = "gene")

# Create from vectors
im4 <- IntervalMap(c(1, 10), ends = c(5, 15), values = c("gene1", "gene2"))
```

---

|                    |                                         |
|--------------------|-----------------------------------------|
| length.IntervalMap | <i>Get the length of an IntervalMap</i> |
|--------------------|-----------------------------------------|

---

## Description

Get the length of an IntervalMap

## Usage

```
## S3 method for class 'IntervalMap'
length(x)
```

## Arguments

|   |                    |
|---|--------------------|
| x | IntervalMap object |
|---|--------------------|

**Value**

Number of intervals in the map

---

|                |                                    |
|----------------|------------------------------------|
| merge_overlaps | <i>Merge overlapping intervals</i> |
|----------------|------------------------------------|

---

**Description**

Returns a new IntervalMap where overlapping intervals are coalesced. The data value of the first interval in coordinate order is kept. Adjacent intervals that do not overlap are not merged. Ties are unspecified.

**Usage**

```
merge_overlaps(x)
```

**Arguments**

x                      IntervalMap object

**Value**

A new IntervalMap with merged intervals

**Examples**

```
im <- IntervalMap(c(1, 5), ends = c(10, 15), values = c("A", "B"))
merged <- merge_overlaps(im)
length(merged) # 1
```

---

|                   |                                     |
|-------------------|-------------------------------------|
| print.IntervalMap | <i>Print method for IntervalMap</i> |
|-------------------|-------------------------------------|

---

**Description**

Print method for IntervalMap

**Usage**

```
## S3 method for class 'IntervalMap'
print(x, ...)
```

**Arguments**

x                      IntervalMap object  
 ...                    Additional arguments (ignored)

**Value**

Invisibly returns the IntervalMap object

---

|         |                                    |
|---------|------------------------------------|
| reserve | <i>Reserve space for intervals</i> |
|---------|------------------------------------|

---

**Description**

Reserve space for intervals

**Usage**

reserve(x, n)

**Arguments**

- x IntervalMap object
- n Number of intervals to reserve space for

**Value**

The IntervalMap object (invisibly)  
The IntervalMap object (invisibly)

**Examples**

```
im <- IntervalMap()  
reserve(im, 1000) # Pre-allocate space for 1000 intervals
```

---

|            |                                                    |
|------------|----------------------------------------------------|
| search_idx | <i>Search for indices of overlapping intervals</i> |
|------------|----------------------------------------------------|

---

**Description**

Search for indices of overlapping intervals

**Usage**

search\_idx(x, start, end)

**Arguments**

- x IntervalMap object
- start Start of query range (inclusive)
- end End of query range (inclusive)

**Value**

Integer vector of 1-based indices

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
build(im)
search_idx.batch(im, 7, 12) # c(1, 2)
```

---

|                  |                                 |
|------------------|---------------------------------|
| search_idx.batch | <i>Batch search for indices</i> |
|------------------|---------------------------------|

---

**Description**

Search for indices of overlapping intervals for multiple query ranges simultaneously. This is more efficient than calling search\_idx multiple times.

**Usage**

```
search_idx.batch(x, starts, ends)
```

**Arguments**

|        |                                                     |
|--------|-----------------------------------------------------|
| x      | IntervalMap object                                  |
| starts | Integer vector of query start positions (inclusive) |
| ends   | Integer vector of query end positions (inclusive)   |

**Value**

List of integer vectors, where each element contains the 1-based indices of intervals that overlap with the corresponding query range

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
add(im, 20, 30, "gene3")
build(im)

# Search multiple ranges at once
result <- search_idx.batch(im, c(7, 25), c(12, 35))
# Returns list(c(1, 2), c(3))
```

---

|              |                                                                               |
|--------------|-------------------------------------------------------------------------------|
| search_items | <i>Search for complete items (start, end, value) of overlapping intervals</i> |
|--------------|-------------------------------------------------------------------------------|

---

**Description**

Search for complete items (start, end, value) of overlapping intervals

**Usage**

```
search_items(x, start, end)
```

**Arguments**

|       |                                  |
|-------|----------------------------------|
| x     | IntervalMap object               |
| start | Start of query range (inclusive) |
| end   | End of query range (inclusive)   |

**Value**

List with start, end, and value vectors

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
build(im)
search_items(im, 7, 12)
```

---

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| search_keys | <i>Search for keys (start, end pairs) of overlapping intervals</i> |
|-------------|--------------------------------------------------------------------|

---

**Description**

Search for keys (start, end pairs) of overlapping intervals

**Usage**

```
search_keys(x, start, end)
```

**Arguments**

|       |                                  |
|-------|----------------------------------|
| x     | IntervalMap object               |
| start | Start of query range (inclusive) |
| end   | End of query range (inclusive)   |

**Value**

List with start and end vectors

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
build(im)
search_keys(im, 7, 12) # list(start=c(1,5), end=c(10,15))
```

---

|              |                                            |
|--------------|--------------------------------------------|
| search_point | <i>Search for values at a single point</i> |
|--------------|--------------------------------------------|

---

**Description**

Returns all values associated with intervals that contain the given point.

**Usage**

```
search_point(x, point)
```

**Arguments**

|       |                               |
|-------|-------------------------------|
| x     | IntervalMap object            |
| point | Position to query (inclusive) |

**Value**

List of values from intervals containing point

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "A")
add(im, 5, 15, "B")
build(im)
search_point(im, 5) # list("A", "B")
```

---

|               |                                                   |
|---------------|---------------------------------------------------|
| search_values | <i>Search for values in overlapping intervals</i> |
|---------------|---------------------------------------------------|

---

**Description**

Search for values in overlapping intervals

**Usage**

```
search_values(x, start, end)
```

**Arguments**

|       |                                  |
|-------|----------------------------------|
| x     | IntervalMap object               |
| start | Start of query range (inclusive) |
| end   | End of query range (inclusive)   |

**Value**

List of values from overlapping intervals

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 5, 15, "gene2")
build(im)
search_values(im, 7, 12) # list("gene1", "gene2")
```

---

|      |                                |
|------|--------------------------------|
| size | <i>Get number of intervals</i> |
|------|--------------------------------|

---

**Description**

Get number of intervals

**Usage**

```
size(x)
```

**Arguments**

|   |                    |
|---|--------------------|
| x | IntervalMap object |
|---|--------------------|

**Value**

Number of intervals

Integer number of intervals in the map

**Examples**

```
im <- IntervalMap()
add(im, 1, 10, "gene1")
add(im, 20, 30, "gene2")
size(im) # Returns 2
```

---

|      |                              |
|------|------------------------------|
| span | <i>Span of all intervals</i> |
|------|------------------------------|

---

**Description**

Returns the smallest interval enclosing every stored interval.

**Usage**

```
span(x)
```

**Arguments**

x                      IntervalMap object

**Value**

A list with start and end, or NULL if the map is empty

**Examples**

```
im <- IntervalMap(c(1, 10), ends = c(5, 20), values = c("A", "B"))
span(im) # list(start = 1, end = 20)
```

---

|           |                                             |
|-----------|---------------------------------------------|
| starts_at | <i>Get start position at specific index</i> |
|-----------|---------------------------------------------|

---

**Description**

Get start position at specific index

**Usage**

```
starts_at(x, idx)
```

**Arguments**

|     |                          |
|-----|--------------------------|
| x   | IntervalMap object       |
| idx | Index position (1-based) |

**Value**

Start position  
Integer start position of the interval

**Examples**

```
im <- IntervalMap()  
add(im, 1, 10, "gene1")  
build(im)  
starts_at(im, 1) # Returns 1
```

---

|                  |                                          |
|------------------|------------------------------------------|
| SuperIntervalMap | <i>Create an indexed genomic subject</i> |
|------------------|------------------------------------------|

---

**Description**

Create an indexed genomic subject

**Usage**

```
SuperIntervalMap(x)
```

**Arguments**

|   |                                                                                                                                            |
|---|--------------------------------------------------------------------------------------------------------------------------------------------|
| x | A GRanges containing subject features, possibly on multiple sequences and strands. Coordinates, names, seqinfo and metadata are preserved. |
|---|--------------------------------------------------------------------------------------------------------------------------------------------|

**Details**

Hits refer to the original subject row order. Strand matching follows GenomicRanges: "\*" matches either strand, and `ignore.strand = TRUE` ignores strand entirely. Ordinary positive-width overlaps use SuperIntervals. Queries involving zero-width ranges, a non-default gap or overlap threshold, or a type other than "any" delegate to GenomicRanges to preserve its semantics. The object stores one index per sequence, with strand filtering performed in C++. Query functions assume the indexes are current; do not alter the slots after construction.

Like IntervalMap, the indexes contain external pointers. Save `as_granges(x)` and reconstruct the index after loading serialized data.

**Value**

A SuperIntervalMap S4 object for use as the subject of `findOverlaps()` and `countOverlaps()`. `as_granges()` retrieves the original GRanges, and `length()` returns its number of features.

**Examples**

```
library(GenomicRanges)
genes <- GRanges(c("chr1", "chr2", "chr1"),
  IRanges(c(20, 1, 1), c(30, 10, 10)),
  strand = c("+", "-", "*"), gene_id = c("C", "B", "A"))
index <- SuperIntervalMap(genes)
query <- GRanges(c("chr1", "chr2"), IRanges(c(5, 5), c(25, 8)),
  strand = c("+", "+"))
findOverlaps(query, index)
countOverlaps(query, index, ignore.strand = TRUE)
as_granges(index)
```

---

SuperIntervalMap-class

*An indexed collection of genomic ranges*

---

**Description**

Index a GRanges once for repeated chromosome- and strand-aware overlap queries. The original ranges, sequence information, names and metadata are retained.

**Usage**

```
## S4 method for signature 'SuperIntervalMap'
length(x)

## S4 method for signature 'SuperIntervalMap'
show(object)
```

**Arguments**

x, object      An indexed genomic map.

**Value**

The S4 class stores indexed GRanges. Its length method returns an integer feature count; show prints a summary and returns invisible NULL.

**Slots**

ranges The original GRanges, in input order.

index Internal per-sequence indexes. Do not edit slots; construct a new object when the subject ranges change.

**See Also**

[SuperIntervalMap\(\)](#)

---

symmetric\_difference    *Symmetric difference of two interval sets*

---

**Description**

Returns a new IntervalMap of regions covered by exactly one of the inputs.

**Usage**

```
symmetric_difference(x, y)
```

**Arguments**

x                      IntervalMap object

y                      IntervalMap object

**Value**

A new IntervalMap representing the symmetric difference

**Examples**

```
a <- IntervalMap(c(1, 20), ends = c(10, 30), values = c("A", "B"))
b <- IntervalMap(c(5, 25), ends = c(15, 35), values = c("C", "D"))
sd <- symmetric_difference(a, b)
```

---

|            |                                   |
|------------|-----------------------------------|
| union_with | <i>Union of two interval sets</i> |
|------------|-----------------------------------|

---

**Description**

Returns a new IntervalMap covering all regions covered by either input. The data value of the first interval in coordinate order is kept. Adjacent intervals that do not overlap are not merged. Ties are unspecified.

**Usage**

```
union_with(x, y)
```

**Arguments**

|   |                    |
|---|--------------------|
| x | IntervalMap object |
| y | IntervalMap object |

**Value**

A new IntervalMap representing the union

**Examples**

```
a <- IntervalMap(c(1, 20), ends = c(10, 30), values = c("A", "B"))
b <- IntervalMap(c(5, 25), ends = c(15, 35), values = c("C", "D"))
u <- union_with(a, b)
```

---

|                  |                                   |
|------------------|-----------------------------------|
| unique_intervals | <i>Remove duplicate intervals</i> |
|------------------|-----------------------------------|

---

**Description**

Returns a new IntervalMap keeping only one interval per distinct (start, end) coordinate pair. One duplicate's data value is kept; the choice among tied duplicates is unspecified.

**Usage**

```
unique_intervals(x)
```

**Arguments**

|   |                    |
|---|--------------------|
| x | IntervalMap object |
|---|--------------------|

**Value**

A new IntervalMap with duplicates removed

**Examples**

```
im <- IntervalMap(c(1, 1, 10), ends = c(5, 5, 15), values = c("A", "B", "C"))
u <- unique_intervals(im)
length(u) # 2
```

# Index

[.IntervalMap, 3

add, 3

as\_granges, 4

at, 4

build, 5

clear, 6

count, 6

count.batch, 7

countOverlaps, GRanges, SuperIntervalMap-method  
(findOverlaps, GRanges, SuperIntervalMap-method), 11

coverage, 8

data\_at, 8

difference, 9

ends\_at, 10

expand, 10

findOverlaps, GRanges, SuperIntervalMap-method, 11

flank, 12

gaps, 13

GenomicRanges::findOverlaps(), 11

has\_overlaps, 13

intersection, 14

IntervalMap, 15

length, SuperIntervalMap-method  
(SuperIntervalMap-class), 25

length.IntervalMap, 16

merge\_overlaps, 17

print.IntervalMap, 17

reserve, 18

search\_idx, 18

search\_idx.batch, 19

search\_items, 20

search\_keys, 20

search\_point, 21

search\_values, 22

show, SuperIntervalMap-method  
(SuperIntervalMap-class), 25

size, 22

span, 23

starts\_at, 24

SuperIntervalMap, 24

SuperIntervalMap(), 26

SuperIntervalMap-class, 25

symmetric\_difference, 26

union\_with, 27

unique\_intervals, 27