




SOFTWARE TOOL ARTICLE

valr: Reproducible genome interval analysis in R [version 1; referees: 2 approved]

Kent A. Riemondy¹, Ryan M. Sheridan², Austin Gillen¹, Yinni Yu²,
Christopher G. Bennett³, Jay R. Hesselberth ^{1,2}

¹RNA Bioscience Initiative, University of Colorado School of Medicine, Aurora, CO, 80045, USA

²Department of Biochemistry and Molecular Genetics, University of Colorado School of Medicine, Aurora, CO, 80045, USA

³ComAnalyzIT LLC, Fort Collins, CO, 80525, USA

v1 First published: 29 Jun 2017, 6:1025 (doi: [10.12688/f1000research.11997.1](https://doi.org/10.12688/f1000research.11997.1))
Latest published: 29 Jun 2017, 6:1025 (doi: [10.12688/f1000research.11997.1](https://doi.org/10.12688/f1000research.11997.1))

Abstract



New tools for reproducible exploratory data analysis of large datasets are important to address the rising size and complexity of genomic data. We developed the valr R package to enable flexible and efficient genomic interval analysis. valr leverages new tools available in the "tidyverse", including dplyr. Benchmarks of valr show it performs similar to BEDtools and can be used for interactive analyses and incorporated into existing analysis pipelines.



This article is included in the [RPackage gateway](#).

Open Peer Review

Referee Status:  

	Invited Referees	
	1	2
version 1 published 29 Jun 2017	 report	 report

- 1 **Robert A. Amezcua**, Yale University, USA
- 2 **Ryan K. Dale**, National Institutes of Health, USA

Discuss this article

Comments (0)

Corresponding author: Jay R. Hesselberth (jay.hesselberth@gmail.com)

Author roles: **Riemony KA:** Conceptualization, Software, Writing – Review & Editing; **Sheridan RM:** Conceptualization, Software, Writing – Review & Editing; **Gillen A:** Software; **Yu Y:** Software, Writing – Review & Editing; **Bennett CG:** Software, Writing – Review & Editing; **Hesselberth JR:** Software, Writing – Original Draft Preparation

Competing interests: No competing interests were disclosed.

How to cite this article: Riemony KA, Sheridan RM, Gillen A *et al.* **valr: Reproducible genome interval analysis in R [version 1; referees: 2 approved]** *F1000Research* 2017, **6**:1025 (doi: [10.12688/f1000research.11997.1](https://doi.org/10.12688/f1000research.11997.1))

Copyright: © 2017 Riemony KA *et al.* This is an open access article distributed under the terms of the [Creative Commons Attribution Licence](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Grant information: This work was supported by the RNA Bioscience Initiative (funded by a Transformational Research Award from the University of Colorado School of Medicine), a grant from the National Institutes of Health (R35 GM119550 to J.H.), the Colorado Office of Economic Development and International Trade (CTGGI 2016- 2096), the BioFrontiers Computing Core at the BioFrontiers Institute, University of Colorado at Boulder and the Intramural Research Program of the National Library of Medicine.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

First published: 29 Jun 2017, **6**:1025 (doi: [10.12688/f1000research.11997.1](https://doi.org/10.12688/f1000research.11997.1))

Introduction

A routine bioinformatic task is the analysis of the relationships between sets of genomic intervals, including the identification of DNA variants within protein coding regions, annotation of regions enriched for nucleic acid binding proteins, and computation of read density within a set of exons. Command-line tools for interval analysis such as `BEDtools`¹ and `BEDOPS`² enable analyses of genome-wide datasets and are key components of analysis pipelines. Analyses with these tools commonly combine processing intervals on the command-line with visualization and statistical analysis in R. However, the need to master both the command-line and R hinders exploratory data analysis, and the development of reproducible research workflows built in the RMarkdown framework.

Existing R packages developed for interval analysis include `IRanges`³, `bedr`⁴, and `GenometriCorr`⁵. `IRanges` is a Bioconductor package that provides interval classes and methods to perform interval arithmetic, and is used by many Bioconductor packages. `bedr` is a CRAN-distributed package that provides wrapper R functions to call the `BEDtools`, `BEDOPS`, and `tabix` command-line utilities, providing out-of-memory support for interval analysis. Finally, `GenometriCorr` provides a set of statistical tests to determine the relationships between interval sets using `IRanges` data structures. These packages provide functionality for processing and statistical inference of interval data, however they require a detailed understanding of S4 classes (`IRanges`) or the installation of external command-line dependencies (`bedr`). Additionally, these packages do not easily integrate with the recent advances provided by the popular `tidyverse` suite of data processing and visualization tools (e.g. `dplyr`, `purrr`, `broom` and `ggplot2`)⁶. We therefore sought to develop a flexible R package for genomic interval arithmetic built to incorporate new R programming, visualization, and interactivity features.

Methods

Implementation

`valr` is an R package that makes extensive use of `dplyr`, a flexible and high-performance framework for data manipulation in R⁷. Additionally, compute intensive functions in `valr` are written in C++ using `Rcpp` to enable fluid interactive analysis of large datasets⁸. Interval intersections and related operations use an interval tree algorithm to efficiently search for overlapping intervals⁹. BED files are imported and handled in R as `data_frame` objects, requiring minimal pre or post-processing to integrate with additional R packages or command-line tools.

Operation

`valr` is distributed as part of the CRAN R package repository and is compatible with Mac OS X, Windows, and major Linux operating systems. Package dependencies and system requirements are documented in the [valr CRAN repository](#).

Use cases

To demonstrate the functionality and utility of `valr`, we present a basic tutorial for using `valr` and additional common use cases for genomic interval analysis.

Basic usage

Input data. `valr` provides a set of functions to read BED, BEDgraph, and VCF formats into R as convenient `tibble` (`tbl`) `data_frame` objects. All `tbls` have `chrom`, `start`, and `end` columns, and `tbls` from multi-column formats have additional pre-determined column names. Standards methods for importing data (e.g. `read.table`, `readr::read_tsv`) are also supported provided the constructed dataframes contain the requisite column names (`chrom`, `start`, `end`). Additionally, `valr` supports connections to remote databases to access the UCSC and Ensembl databases via the `db_ucsc` and `db_ensembl` functions.

```
library(valr)
# function to retrieve path to example data
bed_filepath <- valr_example("3fields.bed.gz")
read_bed(bed_filepath)
#> # A tibble: 10 x 3
#>   chrom  start  end
#>   <chr> <int> <int>
#> 1  chr1  11873  14409
#> 2  chr1  14361  19759
#> 3  chr1  14406  29370
#> 4  chr1  34610  36081
#> 5  chr1  69090  70008
#> 6  chr1 134772 140566
```

```

#> 7 chr1 321083 321115
#> 8 chr1 321145 321207
#> 9 chr1 322036 326938
#> 10 chr1 327545 328439

#using URL
read_bed("https://github.com/rnabioco/valr/raw/master/inst/extdata/3fields.bed.gz")
#> # A tibble: 10 x 3
#>   chrom start end
#>   <chr> <int> <int>
#> 1 chr1 11873 14409
#> 2 chr1 14361 19759
#> 3 chr1 14406 29370
#> 4 chr1 34610 36081
#> 5 chr1 69090 70008
#> 6 chr1 134772 140566
#> 7 chr1 321083 321115
#> 8 chr1 321145 321207
#> 9 chr1 322036 326938
#> 10 chr1 327545 328439

```

Example of combining valr tools. The functions in `valr` have similar names to their `BEDtools` counterparts, and so will be familiar to users of the `BEDtools` suite. Also, similar to `pybedtools`¹⁰, a python wrapper for `BEDtools`, `valr` has a terse syntax. For example, shown below is a demonstration of how to find all intergenic SNPs within 1 kilobase of genes using `valr`. The BED files used in the following examples are described in the Data Availability section.

```

library(dplyr)

snps <- read_bed(valr_example("hg19.snps147.chr22.bed.gz"), n_fields = 6)
genes <- read_bed(valr_example("genes.hg19.chr22.bed.gz"), n_fields = 6)

# find snps in intergenic regions
intergenic <- bed_subtract(snps, genes)
# distance from intergenic snps to nearest gene
nearby <- bed_closest(intergenic, genes)

nearby %>%
  select(starts_with("name"), .overlap, .dist) %>%
  filter(abs(.dist) < 1000)
#> # A tibble: 285 x 4
#>   name.x name.y .overlap .dist
#>   <chr> <chr> <int> <int>
#> 1 rs2261631 P704P 0 -267
#> 2 rs570770556 POTEH 0 -912
#> 3 rs538163832 POTEH 0 -952
#> 4 rs9606135 TPTEP1 0 -421
#> 5 rs11912392 ANKRD62P1-PARP4P3 0 104
#> 6 rs8136454 BC038197 0 355
#> 7 rs5992556 XKR3 0 -455
#> 8 rs114101676 GAB4 0 473
#> 9 rs62236167 CECR7 0 261
#> 10 rs5747023 CECR1 0 -386
#> # ... with 275 more rows

```

Visual documentation. By conducting interval arithmetic entirely in R, `valr` is also an effective teaching tool for introducing interval analysis to early-stage analysts without requiring familiarity with both command-line tools and R. To aid in demonstrating the interval operations available in `valr`, we developed the `bed_glyph()` tool which produces plots demonstrating the input and output of operations in `valr` in a manner similar to those found in the `BEDtools` documentation. Shown below is the code required to produce glyphs displaying the results of intersecting `x` and `y` intervals with `bed_intersect()`, and the result of merging `x` intervals with `bed_merge()` (Figure 1).

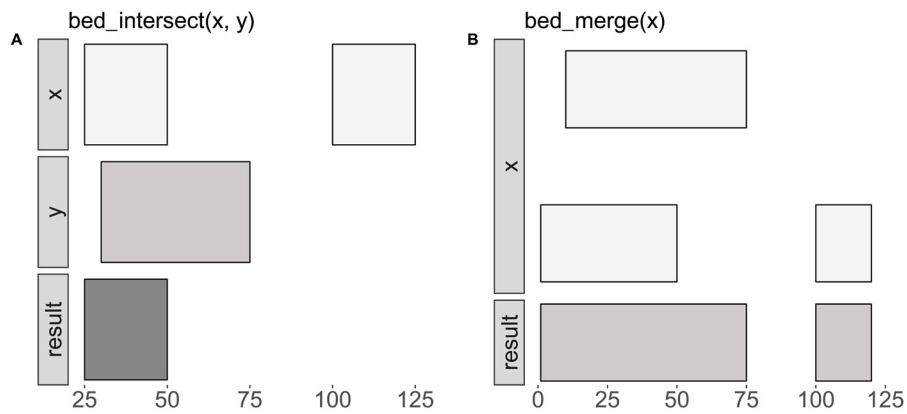


Figure 1. Visualizing interval operations in `valr` with `bed_glyph()`.

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 25,     50,
  "chr1", 100,    125
)

y <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 30,     75
)

bed_glyph(bed_intersect(x, y))
```

And this glyph illustrates `bed_merge()`:

```
x <- tibble::tribble(
  ~chrom, ~start, ~end,
  "chr1", 1,      50,
  "chr1", 10,     75,
  "chr1", 100,   120
)

bed_glyph(bed_merge(x))
```

Grouping data. The `group_by` function in `dplyr` can be used to execute functions on subsets of single and multiple `data_frames`. Functions in `valr` leverage grouping to enable a variety of comparisons. For example, intervals can be grouped by `strand` to perform comparisons among intervals on the same strand.

```
x <- tibble::tribble(
  ~chrom, ~start, ~end, ~strand,
  "chr1", 1,      100, "+",
  "chr1", 50,    150, "+",
  "chr2", 100,   200, "-"
)
```

```

y <- tibble::tribble(
  ~chrom, ~start, ~end, ~strand,
  "chr1", 50, 125, "+",
  "chr1", 50, 150, "-",
  "chr2", 50, 150, "+"
)

# intersect tbls by strand
x <- group_by(x, strand)
y <- group_by(y, strand)

bed_intersect(x, y)
#> # A tibble: 2 x 8
#>   chrom start.x end.x strand.x start.y end.y strand.y .overlap
#>   <chr>   <dbl> <dbl>   <chr>   <dbl> <dbl>   <chr>   <int>
#> 1 chr1     1   100     +     50  125     +     50
#> 2 chr1    50   150     +     50  125     +     75

```

Comparisons between intervals on opposite strands are done using the `flip_strands()` function:

```

x <- group_by(x, strand)

y <- flip_strands(y)
y <- group_by(y, strand)

bed_intersect(x, y)
#> # A tibble: 3 x 8
#>   chrom start.x end.x strand.x start.y end.y strand.y .overlap
#>   <chr>   <dbl> <dbl>   <chr>   <dbl> <dbl>   <chr>   <int>
#> 1 chr2    100  200     -     50  150     -     50
#> 2 chr1     1   100     +     50  150     +     50
#> 3 chr1    50   150     +     50  150     +    100

```

Both single set (e.g. `bed_merge()`) and multi set operations will respect groupings in the input intervals.

Column specification. Columns in BEDtools are referred to by position:

```
# calculate the mean of column 6 for intervals in 'b' that overlap with 'a'
bedtools map -a a.bed -b b.bed -c 6 -o mean
```

In `valr`, columns are referred to by name and can be used in multiple name/value expressions for summaries.

```
# calculate the mean and variance for a 'value' column
bed_map(a, b, .mean = mean(value), .var = var(value))

# report concatenated and max values for merged intervals
bed_merge(a, .concat = concat(value), .max = max(value))
```

API. The major functions available in `valr` are shown in [Table 1](#).

Table 1. An overview of major functions available in valr.

Function Name	Purpose
Reading Data	
read_bed	Read BED files
read_bedgraph	Read bedGraph files
read_narrowpeak	Read narrowPeak files
read_broadpeak	Read broadPeak files
Interval Transformation	
bed_slop	Expand interval coordinates
bed_shift	Shift interval coordinates
bed_flank	Create flanking intervals
bed_merge	Merge overlapping intervals
bed_cluster	Identify (but not merge) overlapping intervals
bed_complement	Create intervals not covered by a query
Interval Comparison	
bed_intersect	Report intersecting intervals from x and y tbls
bed_map	Apply functions to selected columns for overlapping intervals
bed_subtract	Remove intervals based on overlaps
bed_window	Find overlapping intervals within a window
bed_closest	Find the closest intervals independent of overlaps
Randomizing intervals	
bed_random	Generate random intervals from an input genome
bed_shuffle	Shuffle the coordinates of input intervals
Interval statistics	
bed_fisher, bed_projection	Calculate significance of overlaps between two sets of intervals
bed_reldist	Quantify relative distances between sets of intervals
bed_absdist	Quantify absolute distances between sets of intervals
bed_jaccard	Quantify extent of overlap between two sets of intervals
Utilities	
bed_glyph	Visualize the actions of valr functions
bound_intervals	Constrain intervals to a genome reference
bed_makewindows	Subdivide intervals
bed12_to_exons	Convert BED12 to BED6 format
interval_spacing	Calculate spacing between intervals
db_ucsc, db_ensembl	Access remote databases

Summarizing interval coverage across genomic features

This demonstration illustrates how to use valr tools to perform a “meta-analysis” of signals relative to genomic features. Here we analyze the distribution of histone marks surrounding transcription start sites, using H3K4Me3 Chip-Seq data from the ENCODE project.

First we load packages and relevant data.

```
bedfile <- valr_example("genes.hg19.chr22.bed.gz")
genomefile <- valr_example("hg19.chrom.sizes.gz")
bgfile <- valr_example("hela.h3k4.chip.bg.gz")
```

```
genes <- read_bed(bedfile, n_fields = 6)
genome <- read_genome(genomefile)
y <- read_bedgraph(bgfile)
```

Then, we generate 1 bp intervals to represent transcription start sites (TSSs). We focus on + strand genes, but – genes are easily accommodated by filtering them and using `bed_makewindows()` with reversed window numbers.

```
# generate 1 bp TSS intervals, "+" strand only
tss <- genes %>%
  filter(strand == "+") %>%
  mutate(end = start + 1)

# 1000 bp up and downstream
region_size <- 1000
# 50 bp windows
win_size <- 50

# add slop to the TSS, break into windows and add a group
x <- tss %>%
  bed_slop(genome, both = region_size) %>%
  bed_makewindows(genome, win_size)

x
#> # A tibble: 13,530 x 7
#>   chrom   start   end   name score strand .win_id
#>   <chr>   <int>   <int>   <chr> <chr>  <chr>   <int>
#> 1 chr22 16161065 16161115 LINC00516 3      +       1
#> 2 chr22 16161115 16161165 LINC00516 3      +       2
#> 3 chr22 16161165 16161215 LINC00516 3      +       3
#> 4 chr22 16161215 16161265 LINC00516 3      +       4
#> 5 chr22 16161265 16161315 LINC00516 3      +       5
#> 6 chr22 16161315 16161365 LINC00516 3      +       6
#> 7 chr22 16161365 16161415 LINC00516 3      +       7
#> 8 chr22 16161415 16161465 LINC00516 3      +       8
#> 9 chr22 16161465 16161515 LINC00516 3      +       9
#> 10 chr22 16161515 16161565 LINC00516 3      +      10
#> # ... with 13,520 more rows
```

Now we use the `.win_id` group with `bed_map()` to calculate a sum by mapping `y` signals onto the intervals in `x`. These data are regrouped by `.win_id` and a summary with mean and sd values is calculated.

```
# map signals to TSS regions and calculate summary statistics.
res <- bed_map(x, y, win_sum = sum(value, na.rm = TRUE)) %>%
  group_by(.win_id) %>%
  summarize(win_mean = mean(win_sum, na.rm = TRUE),
            win_sd = sd(win_sum, na.rm = TRUE))

res
#> # A tibble: 41 x 3
#>   .win_id win_mean   win_sd
#>   <int>   <dbl>   <dbl>
#> 1     1 100.8974  85.83423
#> 2     2 110.6829  81.13521
#> 3     3 122.9070  99.09635
#> 4     4 116.2800  96.30098
#> 5     5 116.3500 102.33773
#> 6     6 124.9048  95.08887
#> 7     7 122.9437  94.39792
#> 8     8 127.5946  91.47407
#> 9     9 130.2051  95.71309
#> 10    10 130.1220  88.82809
#> # ... with 31 more rows
```

Finally, these summary statistics are used to construct a plot that illustrates histone density surrounding TSSs (Figure 2).

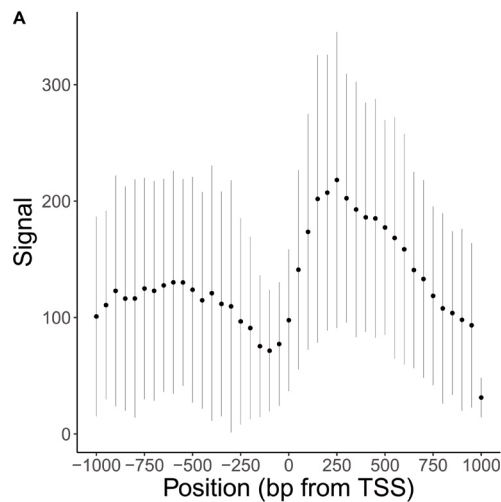


Figure 2. Meta-analysis of signals relative to genomic features with valr. (A) Summarized coverage of human H3K4Me3 Chip-Seq coverage across positive strand transcription start sites on chromosome 22. Data presented +/- SD.

```
library(ggplot2)

x_labels <- seq(-region_size, region_size, by = win_size * 5)
x_breaks <- seq(1, 41, by = 5)

sd_limits <- aes(ymax = win_mean + win_sd, ymin = win_mean - win_sd)

p <- ggplot(res, aes(x = .win_id, y = win_mean)) +
  geom_point(size = 0.25) + geom_pointrange(sd_limits, size = 0.1) +
  scale_x_continuous(labels = x_labels, breaks = x_breaks) +
  xlab("Position (bp from TSS)") + ylab("Signal") +
  theme_classic()
```

Interval statistics

Estimates of significance for interval overlaps can be obtained by combining `bed_shuffle()`, `bed_random()` and the `sample_` functions from `dplyr` with interval statistics in `valr`.

Here, we examine the extent of overlap of repeat classes (repeatmasker track obtained from the UCSC genome browser) with exons in the human genome (hg19 build, on chr22 only, for simplicity) using the jaccard similarity index. `bed_jaccard()` implements the jaccard test to examine the similarity between two sets of genomic intervals. Using `bed_shuffle()` and `replicate()` we generate a data_frame containing 100 sets of randomly selected intervals then calculate the jaccard index for each set against the repeat intervals to generate a null-distribution of jaccard scores. Finally, an empirical p-value is calculated from the null-distribution.

```
library(tidyverse)

repeats <- read_bed(valr_example("hg19.rmsk.chr22.bed.gz"), n_fields = 6)
genome <- read_genome(valr_example("hg19.chrom.sizes.gz"))
genes <- read_bed12(valr_example("hg19.refGene.chr22.bed.gz"))
# convert bed12 to bed with exons
exons <- bed12_to_exons(genes)

# function to repeat interval shuffling
shuffle_intervals <- function(n, .data, genome) {
  replicate(n, bed_shuffle(.data, genome), simplify = FALSE) %>%
    bind_rows(.id = "rep") %>%
    group_by(rep) %>% nest()
}
```

```

nreps <- 100
shuffled <- shuffle_intervals(n = nreps, repeats, genome) %>%
  mutate(jaccard = data %>%
    map床_jaccard, repeats) %>%
    map_dbl("jaccard"))
shuffled
#> # A tibble: 100 x 3
#>   rep      data      jaccard
#>   <chr>    <list>    <dbl>
#> 1     1 <tibble [10,000 x 6]> 0.0003388967
#> 2     2 <tibble [10,000 x 6]> 0.0004965988
#> 3     3 <tibble [10,000 x 6]> 0.0002974843
#> 4     4 <tibble [10,000 x 6]> 0.0006899870
#> 5     5 <tibble [10,000 x 6]> 0.0004678412
#> 6     6 <tibble [10,000 x 6]> 0.0001726937
#> 7     7 <tibble [10,000 x 6]> 0.0004694941
#> 8     8 <tibble [10,000 x 6]> 0.0004660410
#> 9     9 <tibble [10,000 x 6]> 0.0006846643
#> 10    10 <tibble [10,000 x 6]> 0.0002143829
#> # ... with 90 more rows

obs <- bed_jaccard(repeats, exons)
obs
#> # A tibble: 1 x 4
#>   len_i  len_u  jaccard  n
#>   <dbl> <dbl>    <dbl> <dbl>
#> 1 112123 4132109 0.02789139 805

pvalue <- sum(shuffled$jaccard >= obs$jaccard) + 1 / (nreps + 1)
pvalue
#> [1] 0.00990099

```

Benchmarking against bedtools

In order to ensure that `valr` performs fast enough to enable interactive analysis, key functionality is implemented in C++. To test the speed of major `valr` functions we generated two `data_frames` containing 1 million randomly selected 1 kilobase intervals derived from the human genome (hg19). Most of the major `valr` functions complete execution in less than 1 second, demonstrating that `valr` can process large interval datasets efficiently (Figure 3A).

We also benchmarked major `valr` functions against corresponding commands in `BEDtools`. `valr` operates on `data_frames` already loaded into RAM, whereas `BEDtools` performs file-reading, processing, and writing. To compare `valr` against `BEDtools` we generated two `BED` files containing 1 million randomly selected 1 kilobase intervals derived from the human genome (hg19). For `valr` functions, we timed reading the table into R (e.g. with `read_bed()`) and performing the respective function. For `BEDtools` commands we timed executing the command with the output written to `/dev/null`. `valr` functions performed similarly or faster than `BEDtools` commands, with the exception of `bed_map` and `bed_fisher` (Figure 3B).

Reproducible reports and interactive visualizations

Command-line tools like `BEDtools` and `bedops` can be incorporated into reproducible workflows (e.g., with `snakemake`¹¹), but it is cumbersome to transition from command-line tools to exploratory analysis and plotting software. RMarkdown documents are plain text files, amenable to version control, which provide an interface to generate feature rich PDF and HTML reports that combine text, executable code, and figures in a single document. `valr` can be used in RMarkdown documents to provide rapid documentation of exploratory data analyses and generate reproducible work-flows for data processing. Moreover, new features in RStudio, such as notebook viewing, and multiple language support enable similar functionality to another popular notebook platform `jupyter` notebooks.

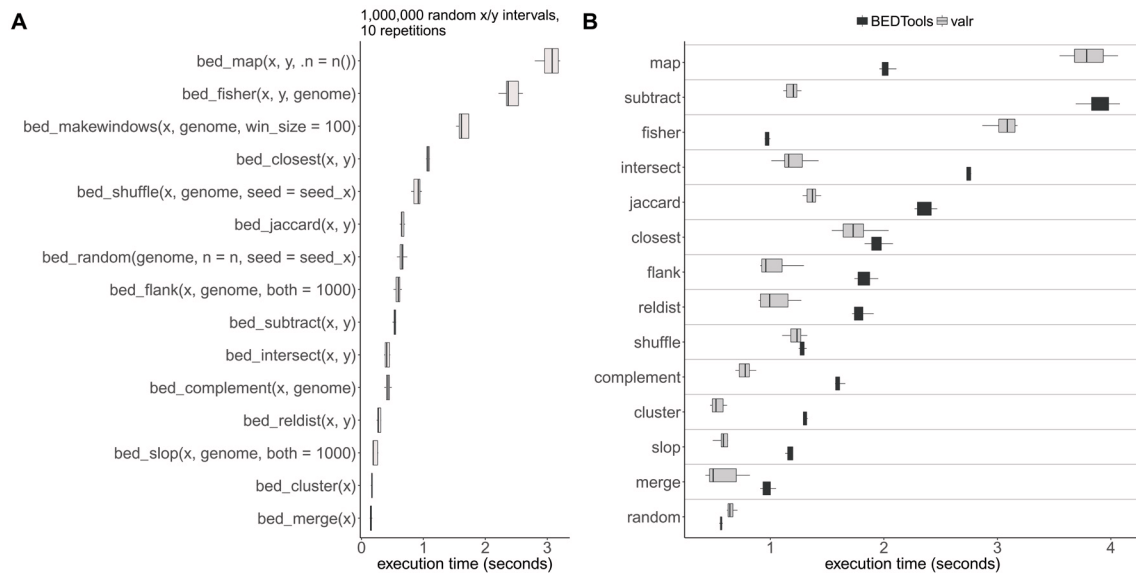


Figure 3. Performance of valr functions. (A) Timings were calculated by performing 10 repetitions of indicated functions on data frames preloaded in R containing 1 million random 1 kilobase x/y intervals generated using `bed_random()`. (B) Timings for executing functions in `BEDTools v2.25.0` or equivalent functions in `valr` using the same interval sets as in (A) written to files. All `BEDTools` function outputs were written to `/dev/null/`, and were timed using `GNU time`. Timings for `valr` functions in (B) include times for reading files using `read_bed()` functions and were timed using the `microbenchmark` package.

Additionally, `valr` seamlessly integrates into R `shiny`¹² applications allowing for complex interactive visualizations relating to genomic interval analyses. We have developed a `shiny` application (available on [Github](#)) that explores ChIP-Seq signal density surrounding transcription start sites and demonstrates the ease of implementing `valr` to power dynamic visualizations.

Summary

`valr` provides a flexible framework for interval arithmetic in R/Rstudio. `valr` functions are written with a simple and terse syntax that promotes flexible interactive analysis. Additionally by providing an easy-to-use interface for interval arithmetic in R, `valr` is also a useful teaching tool to introduce the analyses necessary to investigate correlations between genomic intervals, without requiring familiarity with the command-line. We envision that `valr` will help researchers quickly and reproducibly analyze genome interval datasets.

Data and software availability

The `valr` package includes external datasets stored in the `inst/extdata/` directory that were used in this manuscript. These datasets were obtained from the ENCODE Project¹³ or the UCSC genome browser¹⁴. BED files were generated by converting the UCSC tables into BED format. BED and BEDgraph data was only kept from chromosome 22, and was subsampled to produce file sizes suitable for submission to the CRAN repository. The original raw data is available from the following sources:

hela.h3k4.chip.bg.gz SRA record: SRR227441, ENCODE identifier: ENCSR000AOF

hg19.refGene.chr22.bed.gz <ftp://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/refGene.txt.gz>

hg19.rmsk.chr22.bed.gz <ftp://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/rmsk.txt.gz>

hg19.chrom.sizes.gz <ftp://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/chromInfo.txt.gz>

genes.hg19.chr22.bed.gz <ftp://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/refGene.txt.gz>

hg19.snps147.chr22.bed.gz <ftp://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/snp147.txt.gz>

valr can be installed via CRAN using `install.packages("valr")`.

valr is maintained at <http://github.com/rnabioco/valr>.

Latest valr source code is available at <http://github.com/rnabioco/valr>.

The latest stable version of source code is at: <https://github.com/rnabioco/valr/archive/v0.3.0.tar.gz>

Archived source code at the time of publication: <http://doi.org/10.5281/zenodo.815403>¹⁵

License: MIT license.

Competing interests

No competing interests were disclosed.

Grant information

This work was supported by the RNA Bioscience Initiative (funded by a Transformational Research Award from the University of Colorado School of Medicine), a grant from the National Institutes of Health (R35 GM119550 to J.H.), the Colorado Office of Economic Development and International Trade (CTGGI 2016-2096), the BioFrontiers Computing Core at the BioFrontiers Institute, University of Colorado at Boulder and the Intramural Research Program of the National Library of Medicine.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgments

This work was in part completed during an NIH sponsored Hackathon hosted by the Biofrontiers Department at the University of Colorado at Boulder.

References

- Quinlan AR, Hall IM: **BEDTools: a flexible suite of utilities for comparing genomic features**. *Bioinformatics*. 2010; **26**(6): 841–842. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Neph S, Kuehn MS, Reynolds AP, et al.: **BEDOPS: high-performance genomic feature operations**. *Bioinformatics*. 2012; **28**(14): 1919–1920. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Lawrence M, Huber W, Pagès H, et al.: **Software for computing and annotating genomic ranges**. *PLoS Comput Biol*. 2013; **9**(8): e1003118. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Haider S, Waggott D, Lalonde E, et al.: **A bedr way of genomic interval processing**. *Source Code Biol Med*. 2016; **11**: 14. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Favorov A, Mularoni L, Cope LM, et al.: **Exploring massive, genome scale datasets with the GenometriCorr package**. *PLoS Comput Biol*. 2012; **8**(5): e1002529. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Wickham H: **tidyverse: Easily Install and Load 'Tidyverse' Packages**. R package version 1.1.1, 2017. [Reference Source](#)
- Wickham H, Francois R: **dplyr: A Grammar of Data Manipulation**. R package version 0.5.0, 2016. [Reference Source](#)
- Eddelbuettel D, François R: **Rcpp: Seamless R and C++ integration**. *J Stat Softw*. 2011; **40**(8): 1–18. [Publisher Full Text](#)
- Cormen TH, Leiserson CE, Rivest RL, et al.: **Introduction to Algorithms**. 2nd Ed. Cambridge (Massachusetts): MIT Press; 2001. [Reference Source](#)
- Dale RK, Pedersen BS, Quinlan AR: **Pybedtools: a flexible Python library for manipulating genomic datasets and annotations**. *Bioinformatics*. 2011; **27**(24): 3423–3424. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Köster J, Rahmann S: **Snakemake—a scalable bioinformatics workflow engine**. *Bioinformatics* 2012; **28**(19): 2520–2522. [PubMed Abstract](#) | [Publisher Full Text](#)
- Chang W, Cheng J, Allaire JJ, et al.: **shiny: Web Application Framework for R**. R package version 1.0.3, 2017. [Reference Source](#)
- ENCODE Project Consortium: **An integrated encyclopedia of DNA elements in the human genome**. *Nature*. 2012; **489**(7414): 57–74. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Rosenbloom KR, Armstrong J, Barber GP, et al.: **The UCSC Genome Browser database: 2015 update**. *Nucleic Acids Res*. 2015; **43**(Database issue): D670–D681. [PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Hesselberth J, kriemo, sheridar, et al.: **rnabioco/valr: Zenodo release**. *Zenodo*. 2017. [Data Source](#)

Open Peer Review

Current Referee Status:  

Version 1

Referee Report 10 July 2017

doi:10.5256/f1000research.12975.r24101



Ryan K. Dale

Laboratory of Cellular and Developmental Biology, National Institute of Diabetes and Digestive and Kidney Diseases, National Institutes of Health, Bethesda, MD, USA

The authors describe a package for manipulating genomic interval data in R using principles from the "tidyverse" for the data structures and API. This sets it apart from existing tools such as GenomicRanges or bedr which have their own ways of storing and manipulating data. As a result, valr should be easier to pick up and integrate with the rest of the R ecosystem, and the "tidyverse" in particular. Illustrative examples give the reader a taste for the package while highlighting the novel features.

In general, this looks to be a very useful tool. The code quality is excellent and it is great to see so many tests including the addition of regression tests as issues are identified.

My comments are very minor:

- Group-by code listing: comment "# intersect tbls by strand" should be "# group tbls by strand"
- Bioconductor might be a more appropriate repository than CRAN
- Description of in-memory usage: I see from the software documentation that BAM and VCF will be supported in the future, and the documentation explicitly mentions that valr operates on data in-memory. The section comparing with BEDTools briefly mentions the in-memory aspect, but it would be helpful to be clearer about memory usage in the manuscript, especially as users attempting to use large BAM files may run out of memory.
- This is just a suggestion for improvement: Over the years, numerous bugs from corner cases have been found and handled in BEDTools. It would greatly increase confidence in the underlying algorithms you have written if there is input/output parity between valr and BEDTools, at least for the tools that overlap the two packages. For example I see some test cases that use input from the BEDTools test suite (e.g., test_cluster.r), but don't check the output. It should be straightforward to check the output against that provided by the BEDTools test suite. Correspondingly it would be good for BEDTools to use valr input/expected output in its test suite.

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Referee Report 10 July 2017

doi:[10.5256/f1000research.12975.r23916](https://doi.org/10.5256/f1000research.12975.r23916)



Robert A. Amezcua

Department of Immunobiology, School of Medicine, Yale University, New Haven, CT, USA

Riemony and colleagues have made a valuable tool, `valr`, available to the general public, one which deserves much merit for bringing modern R idioms from the `tidyverse` into the world of genomic research. As a fellow bioinformatician who has struggled with the idiosyncrasies of the aforementioned tools for interval manipulation in R, `valr` addresses many of the usability issues associated with these legacy methods by fundamentally altering the user experience. Consequently, there are two main advantages to adopting `valr` for interval manipulation in R: ease of writing, and ease of reading, code. Thus, this referee wholeheartedly endorses `valr`, and hopes to see more work that brings many of the `tidyverse` philosophies over to working with genomics in R.

Nonetheless, there is some room for improvement of the associated manuscript to better help explain the philosophy and usage of `valr`, and its place amongst the many tools for the manipulation of genomic intervals.

Firstly, in the introduction, it is mentioned that there exist `IRanges` methods that utilize the S4 convention, whereas `valr` utilizes a less formal schema where 3 columns, `chrom`, `start`, `end`, are present in the `data_frame` object. Indeed, it may be of use to expand upon such design choices that were made, and what advantages/disadvantages are made in using this less formal schema, and any other highly pertinent choices that affect user experience. In addition, one line mentions integration with other `tidyverse` tools, and should expand upon this with either one to a few specific examples or explain this point in more detail. Additionally, it should also be pointed out how `valr` builds upon these existing toolkits, and either expands upon/adopts their conventions. One way might be to create a table comparing

functions between ``valr`/bedtools/GenomicRanges` might be helpful for a reader to see that the toolkit will be easily adoptable. Indeed, its mentioned that the syntax is similar to bedtools in the use cases, and might be good to mention in the introduction as well. Thus, an expanded introduction/additional section explaining the uniqueness of ``valr`` would help to better "sell" when one should use ``valr`` and why.

In performing benchmarking, it would be useful to include one or two leading R tools, such as `GenomicRanges`, into the calculations, as this is likely how many R programmers currently perform interval manipulations natively in R, and I suspect would likely show an impressive performance improvement by relation.

``valr`` presents an exciting new development in the R+Genomics realm, and this referee is hopeful that this sort of development helps fuel further ``tidyomics`` tools for R bound together by a cohesive philosophy, great user experience, and pointed utility.

Is the rationale for developing the new software tool clearly explained?

Partly

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Referee Expertise: Computational immunology

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.
