

UCLA

UCLA Previously Published Works

Title

A bedr way of genomic interval processing.

Permalink

<https://escholarship.org/uc/item/6n31x67w>

Journal

Source code for biology and medicine, 11(1)

ISSN

1751-0473

Authors

Haider, Syed
Waggott, Daryl
Lalonde, Emilie
et al.

Publication Date

2016

DOI

10.1186/s13029-016-0059-5

Peer reviewed

SOFTWARE

Open Access



A bedr way of genomic interval processing

Syed Haider^{1†}, Daryl Waggott^{1†}, Emilie Lalonde^{1,2}, Clement Fung¹, Fei-Fei Liu^{2,3} and Paul C. Boutros^{1,2*}

Abstract

Background: Next-generation sequencing is making it critical to robustly and rapidly handle genomic ranges within standard pipelines. Standard use-cases include annotating sequence ranges with gene or other genomic annotation, merging multiple experiments together and subsequently quantifying and visualizing the overlap. The most widely-used tools for these tasks work at the command-line (e.g. BEDTools) and the small number of available R packages are either slow or have distinct semantics and features from command-line interfaces.

Results: To provide a robust R-based interface to standard command-line tools for genomic coordinate manipulation, we created bedr. This open-source R package can use either BEDTools or BEDOPS as a back-end and performs data-manipulation extremely quickly, creating R data structures that can be readily interfaced with existing computational pipelines. It includes data-visualization capabilities and a number of data-access functions that interface with standard databases like UCSC and COSMIC.

Conclusions: bedr package provides an open source solution to enable genomic interval data manipulation and restructuring in R programming language which is commonly used in bioinformatics, and therefore would be useful to bioinformaticians and genomic researchers.

Keywords: Genomic intervals, BED format, Sequence algebra, Data integration

Background

With the advent of high-throughput sequencing technologies, data scientists are facing immense challenges in large-scale sequence analysis and in integrating genomic annotations. For instance, comparing new experiments with previously published datasets, translating genomic coordinates between different assemblies of an organism as well as finding cross-species orthologues are some of the common use-cases in basic science experiments. To assist these tasks genomic features are routinely represented and shared using Browser Extensible Display (BED; [1]), Distributed Annotation System (DAS; [2]), General Feature Format (GFF), Gene Transfer Format (GTF) and Variant Call Format (VCF). These all enable cross-sectional analysis of genomic studies across multiple programming languages, thereby enabling seamless data-integration.

R is the *de facto* standard for statistical analysis and visualization in computational biology [3] for both exploratory prototyping and rigorous production pipelines. To this end R has adopted several packages, such as GenomicRanges and IRanges that expressly deal with genomic intervals [4]. Albeit powerful, these existing tools require understanding of bespoke data structures and classes/objects. To address these issues we implemented a formal BED-operations framework called bedr, which is an R API offering utility functions implementing commonly used genomic operations as well as offering a formal R interface to interact with BEDTools and BEDOPS. bedr is fully compatible with the ubiquitous BED tools [5, 6] paradigm and integrates seamlessly with R-based work-flows.

Implementation

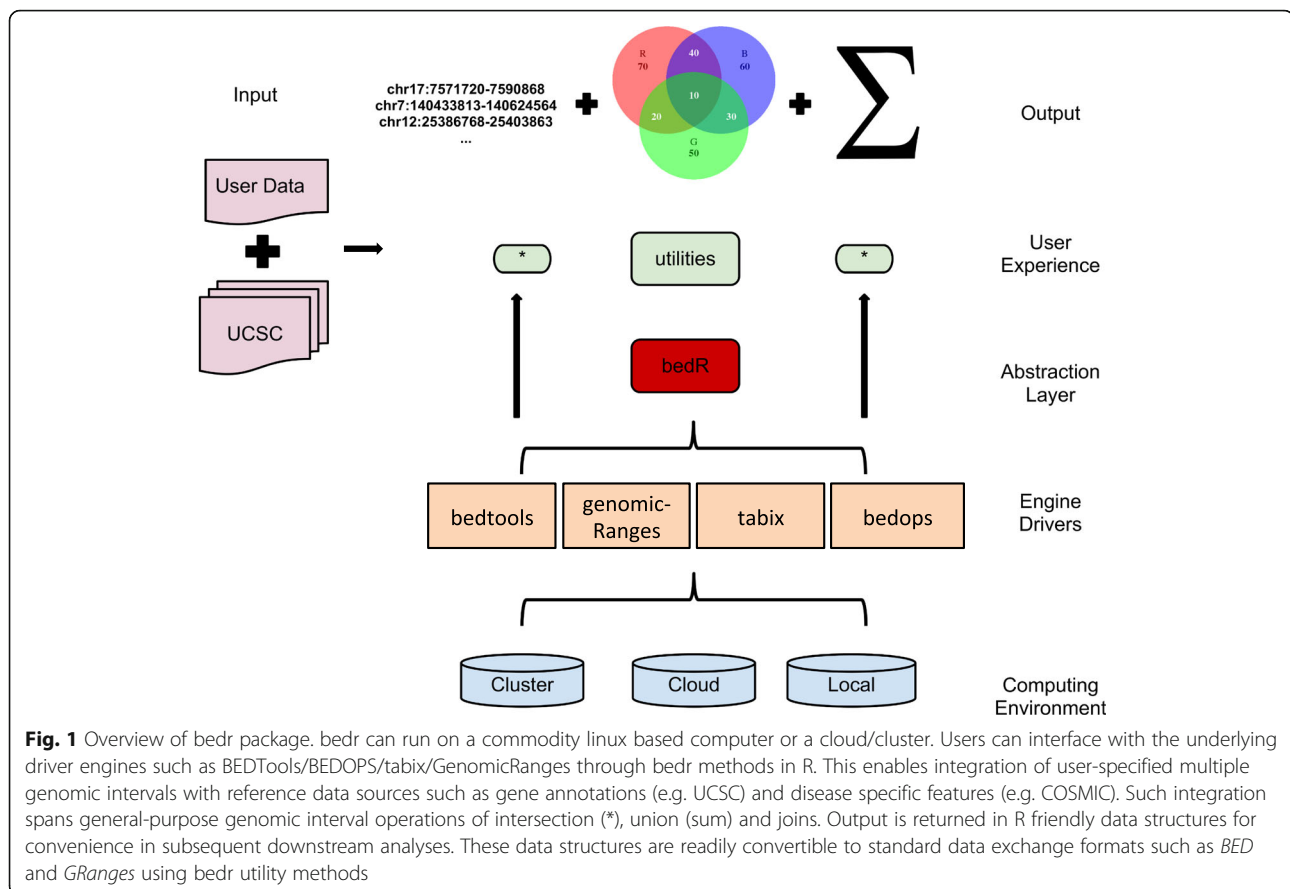
bedr is implemented in R and supports the two main BED engines: BEDTools and BEDOPS [5, 6]. It works on Unix-like operating systems including Mac OSX. A high-level conceptual overview of the package usage is shown in Fig. 1. bedr functions run on any computing platform (commodity computer, cluster or cloud), and facilitates integration of multiple data sources (e.g. gene

* Correspondence: Paul.Boutros@oicr.on.ca

†Equal contributors

¹Informatics and Biocomputing Platform, Ontario Institute for Cancer Research, Toronto M5G 0A3, Canada

²Departments of Radiation Oncology, Pharmacology & Toxicology, and Medical Biophysics, University of Toronto, Toronto M5G 2M9, Canada
Full list of author information is available at the end of the article



annotations) and tools (e.g. BEDTools) with user-specified genomic coordinates, yielding fast results in R data structures such as *data.frame* and *list*. All API calls are written in R, which internally calls BEDTools and/or BEDOPS utilities in the native format using system commands. Further, bedR data can be readily converted to *BED* [1] and *GRanges* [4] objects for wider interoperability.

Results and discussion

The primary input to most bedR methods is a regions object, which is represented as either an R *vector* of multiple region strings as illustrated below or a *data.frame* of regions with three columns: *chr*, *start*, and *end*. The regions object returned by various bedR methods matches the input format; *vector* or *data.frame*. Here we briefly summarize a subset of key bedR functionalities. For further details on a range of bedR utilities, please see package's help and vignettes for detailed examples and workflows.

Sort & merge

This functionality enables sorting of genomic regions in both natural and lexicographical order using R, unix,

BEDTools and BEDOPS engines. Following examples demonstrate the usage of these engines:

```
regions <- get.example.regions()
region <- regions[[1]]
bedr.sort.region(
  x = region,
  engine = "unix",
  method = "natural"
)
bedr.sort.region(
  x = region,
  engine = "R",
  method = "lexicographical"
)
bedr.sort.region(
  x = region,
  engine = "bedtools"
)
bedr.sort.region(
  x = region,
  engine = "bedops"
)
```

The above code will generate the following outputs of sorted regions:

```
# natural sort (unix)
"chr1:10-100" "chr1:101-200"
"chr1:200-210" "chr1:211-212"
"chr2:10-50" "chr2:40-60"
"chr10:50-100" "chr20:1-5"

# lexicographical sort (R)
"chr1:10-100" "chr1:101-200"
"chr1:200-210" "chr1:211-212"
"chr10:50-100" "chr2:10-50"
"chr2:40-60" "chr20:1-5"

# lexicographical sort (bedtools)
"chr1:10-100" "chr1:101-200"
"chr1:200-210" "chr1:211-212"
"chr10:50-100" "chr2:10-50"
"chr2:40-60" "chr20:1-5"

# lexicographical sort (bedops)
"chr1:10-100" "chr1:101-200"
"chr1:200-210" "chr1:211-212"
"chr10:50-100" "chr2:10-50"
"chr2:40-60" "chr20:1-5"
```

As shown above, various types of sorting results are presented in a similar R data structures regardless of which sorting engine is used (unix, R, bedtools or bedops) and their respective output style. Also, BEDTools and BEDOPS do not support natural sorting, and if method = "natural" is requested with these two engines, bedr automatically defaults to using engine = "unix" of "R" to perform sorting. Note, sorting of large number of regions through R will be slow and may also result in high memory overhead.

Much of the command-line interaction with BEDTools and BEDOPS is performed through temporary files followed by efficient piping/parsing of the output straight into R data structures. This ensures that memory intensive sorting tasks (or any other genomic operations discussed below) are managed by the optimized engines such as (BEDTools or BEDOPS), and therefore memory operations in R are limited to subsequent parsing of output.

In addition to sort operations, bedr also supports identification of overlapping regions which can be collapsed to avoid downstream analytical challenges such as many:many join results (Fig. 2), e.g.

```
bedr.merge.region(x = region)
```

The above code will generate the following output of merged regions:

```
"chr1:10-100" "chr1:101-210"
"chr1:211-212" "chr10:50-100"
"chr2:10-60" "chr20:1-5"
```

Sort and merge can be combined into one step given they are generally run as a tandem preprocessing step:

```
bedr.snm.region(x = region)
```

The above code will generate the following vector output of sorted and merged regions:

```
"chr1:10-100" "chr1:101-210"
"chr1:211-212" "chr10:50-100"
"chr2:10-60" "chr20:1-5"
```

Join

This functionality enables joining two region-based datasets using intervals as an index or primary key. The output is left outer join with respect to the first regions object (Fig. 2), e.g.

```
regions.a <- bedr.merge.region(
  x = regions[[1]]
)
regions.b <- bedr.merge.region(
  x = regions[[2]]
)
regions.c <- bedr.merge.region(
  x = regions[[4]]
)
bedr.join.region(
  x = regions.a,
  y = regions.b
)
```

The above code will generate the following output, containing regions of regions.a in the first column, while any overlapping regions from regions.b are listed in columns 2 to 4 (chr, start, end). Regions in regions.a with no overlap are encoded as: . and -1

	index	V4	V5	V6
1	chr1:10-100	.	-1	-1
2	chr1:101-210	chr1	111	250
3	chr1:211-212	chr1	111	250
4	chr10:50-100	.	-1	-1
5	chr2:10-60	chr2	40	60
6	chr20:1-5	.	-1	-1

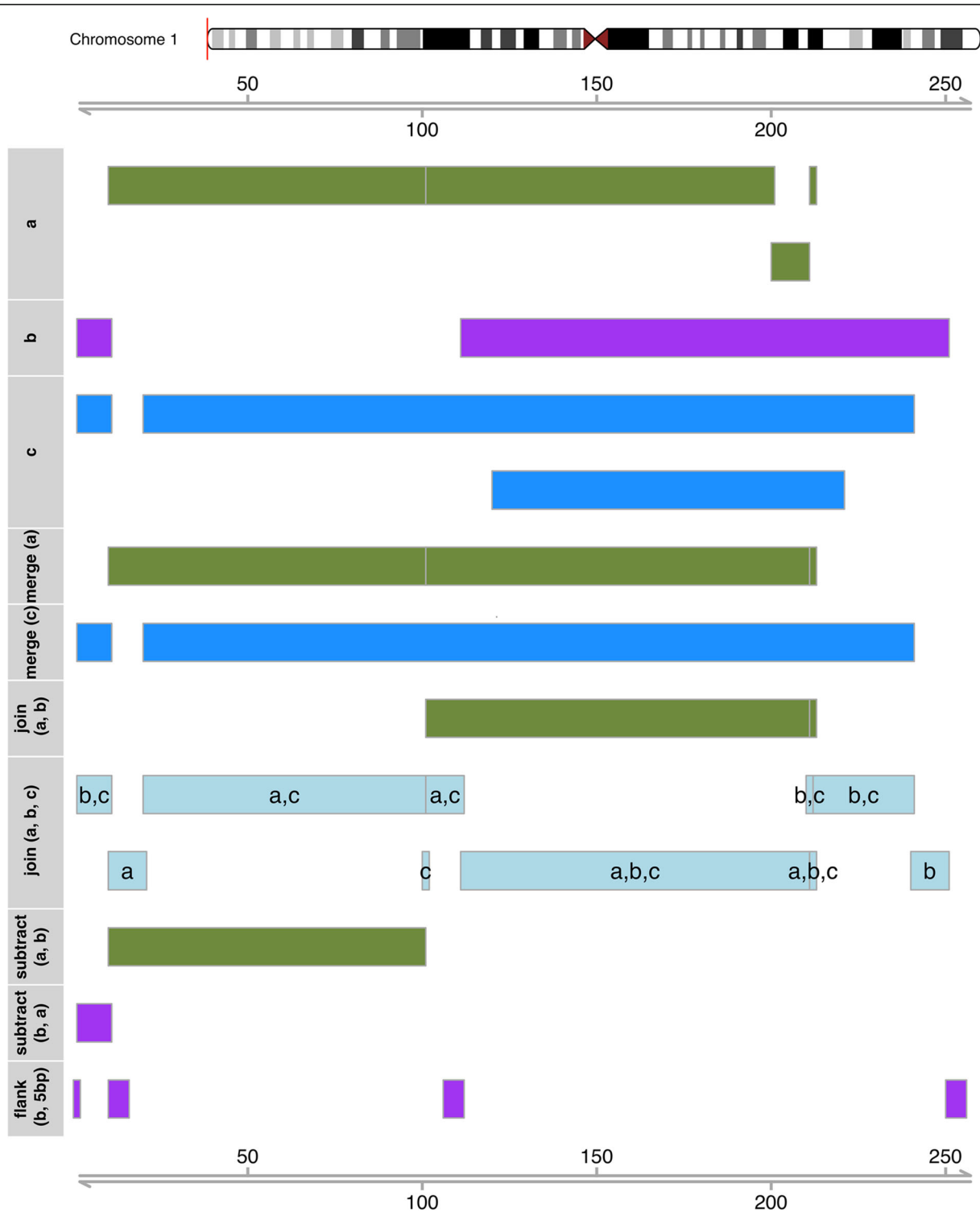


Fig. 2 Illustration of key `bedr` operations. `bedr` regions objects represent a collection of sub-regions specified as R *vector* or *data.frame*. Three partially overlapping example regions (`a`, `b` and `c`) located at the beginning of human chromosome 1 (red mark on ideogram, 1-250 bp) are shown here. Vertical gray separators between sub-regions indicate regions that are 1 base pair apart. Overlapping regions can be merged, joined, subtracted resulting in new regions objects as shown here. Associated source code snippets are documented in the Results section. Regions object `flank(b, 5bp)` exemplifies `bedr` utility `flank.regions` creating flanking (up and/or downstream) regions of a specified length; ± 5 bp in the example shown here

Similarly, another `bedr` function `bedr.join.multiple.region()` supports merging of multiple sets of regions (Fig. 2), e.g.

```
bedr.join.multiple.region(
  x = list(
    a = regions.a,
    b = regions.b,
    c = regions.c
  )
)
```

The above code will generate the output `data.frame` shown below. The table lists all the sub-regions and their presence across the three sets of region objects (`regions.a`, `regions.b`, and `regions.c`) passed to the function. For instance, sub-region `chr1:1-10` (column: index) overlaps with 2 region objects (b and c). This presence is shown as comma separated list in 'names' column as well as a truth table in the subsequent columns. The number of columns representing the truth table will match the number of region objects passed to the function `bedr.join.multiple.region()`.

	index	n.overlaps	names	a	b	c
1	chr1:1-10	2	b,c	0	1	1
2	chr1:10-20	1	a	1	0	0
3	chr1:20-100	2	a,c	1	0	1
4	chr1:100-101	1	c	0	0	1
5	chr1:101-111	2	a,c	1	0	1
6	chr1:111-210	3	a,b,c	1	1	1
7	chr1:210-211	2	b,c	0	1	1
8	chr1:211-212	3	a,b,c	1	1	1
9	chr1:212-240	2	b,c	0	1	1
10	chr1:240-250	1	b	0	1	0
11	chr1:2000-2010	1	b	0	1	0
12	chr10:50-100	1	a	1	0	0
13	chr10:100-110	1	b	0	1	0
14	chr10:110-150	2	b,c	0	1	1
15	chr2:1-5	2	b,c	0	1	1
16	chr2:5-10	1	c	0	0	1
17	chr2:10-20	2	a,c	1	0	1
18	chr2:20-30	1	a	1	0	0
19	chr2:30-40	2	a,c	1	0	1
20	chr2:40-60	3	a,b,c	1	1	1
21	chr20:1-5	1	a	1	0	0
22	chr20:6-7	1	b	0	1	0
23	chr20:7-10	2	b,c	0	1	1
24	chr20:10-12	1	c	0	0	1

Subtract and intersect

The `subtract` utility identifies regions exclusive to first set of regions, and `intersect` function identifies sub-regions of first set which overlap with the second set of regions (Fig. 2), e.g.

```
bedr.subtract.region(
  x = regions.a,
  y = regions.b
)
```

The above code will generate the following output which lists the sub-regions exclusive to `regions.a`:

```
"chr1:10-100" "chr10:50-100"
"chr20:1-5"
```

`Intersect` utility makes use of `bedr.join.region()` and finds regions in the second set which overlap with the regions in the first set. An example is shown in the Results section "Join". Similarly `in.region(x = regions.a, y = regions.b)` and its R style convenience operator `%in.region%` can be used to test (logical) presence of overlapping regions, e.g.

```
in.region(
  x = regions.a,
  y = regions.b
)
```

```
FALSE TRUE TRUE FALSE TRUE FALSE
```

`bedr` also provides an interface to find overlapping regions using *Tabix* [7]. This can be done using the following `bedr` call:

```
regions.d <- c(
  "1:1000-100000",
  "1:1000000-1100000"
)

cosmic.vcf.example <- system.file(
  "extdata/CosmicCodingMuts_v66_
  20130725_ex.vcf.gz",
  package = "bedr"
)

head(
  tabix(
    region = regions.d,
    file.name = cosmic.vcf.example,
    check.chr = FALSE
  )
)
```

which identifies regions overlapping with COSMIC coding mutations file resulting in the following *data.frame* (only first six rows are shown below):

CHROM	POS	ID	REF	ALT	QUAL	FILTER
1	1	69345	COSM911918	C	A	NA <NA>
2	1	69523	COSM426644	G	T	NA <NA>
3	1	69538	COSM75742	G	A	NA <NA>
4	1	69539	COSM1343690	T	C	NA <NA>
5	1	69540	COSM1560546	G	T	NA <NA>
6	1	69569	COSM1599955	T	C	NA <NA>

INFO

```

1 GENE=OR4F5;STRAND=+;CDS=c.255C>A;AA=p.I85I;CNT=1
2 GENE=OR4F5;STRAND=+;CDS=c.433G>T;AA=p.G145C;CNT=1
3 GENE=OR4F5;STRAND=+;CDS=c.448G>A;AA=p.V150M;CNT=1
4 GENE=OR4F5;STRAND=+;CDS=c.449T>C;AA=p.V150A;CNT=1
5 GENE=OR4F5;STRAND=+;CDS=c.450G>T;AA=p.V150V;CNT=1
6 GENE=OR4F5;STRAND=+;CDS=c.479T>C;AA=p.L160P;CNT=2

```

Third-party compatibility

Given that `bedr` can process regions data as R's *vector* as well as *data.frame* data structure, it is easily transformable to other third-party sequence and region objects. For instance, `bedr` provides a utility adaptor to convert regions into *BED data.frame* as shown below:

```

regions.a.bed <- convert2bed(
  x = regions.a
)

```

which can be further converted to a widely compatible *GRanges* [4] object, as shown below:

```

library("GenomicRanges")
makeGRangesFromDataFrame(
  df = regions.a.bed
)

```

The above code will create a *GRanges* object shown in the output below, which can be further customized/extended with additional annotations such as strand and genomic feature names.

GRanges object with 6 ranges
and 0 metadata columns:

	seqnames	ranges	strand
	<Rle>	<IRanges>	<Rle>
[1]	chr1	[10, 100]	*
[2]	chr1	[101, 210]	*
[3]	chr1	[211, 212]	*
[4]	chr10	[50, 100]	*
[5]	chr2	[10, 60]	*
[6]	chr20	[1, 5]	*

seqinfo: 4 sequences from an
unspecified genome; no seqlengths

To perform feature meta-analysis and annotation retrieval/conversion (see example workflow in Additional file 1), `bedr` facilitates downloads from UCSC [8], COSMIC [9] and HUGO [10] including reference genome annotations, repeat sequences, black lists and disease candidate features. Also, `bedr` has a fully integrated unit-testing framework allowing users to verify integrity of `bedr` functions when using customized development or installations.

Visualization

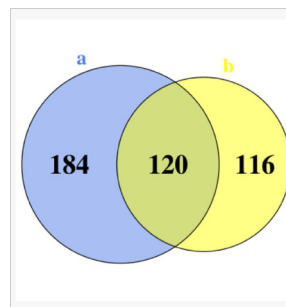
For results of common operations such as intersect, Venn diagrams of overlapping features between 2 to 5 sets of regions (2- to 5-way Venn diagrams) can be generated automatically [11]. The overlap criterion can be defined in a number of ways including unique intervals, gene length or user-specified size as a fraction of sub-region's length, e.g.

```

bedr.plot.region(
  input = list(
    a = regions.a,
    b = regions.b
  ),
  feature = "bp",
  fraction.overlap = 0.1
)

```

The above code will generate a base pair level overlap of sequence objects `regions.a` and `regions.b`, and show the results as a Venn diagram highlighting lengths of exclusive and overlapping regions as shown below:



Further, `bedr` output is ideally suited for alternative complex set visualization tools such as *UpSetR* [12] and *Gviz* [13].

Conclusions

We created `bedr`; an R package to support genomic operations using the *BEDTools* [6] and *BEDOPS* [5] engines. `bedr` implements an API in R which offers a number of utility functions such as intersect, merge, sort and plotting of genomic intervals as well as provides a unified interface to *BEDTools* and *BEDOPS*. These functions are efficient, powerful and perform complex feature annotations and cross-sectional operations on genomic regions. Given that `bedr` supports two well-established genomic engines, its output is comparable to the native output of these tools, however in R data structures. These features of `bedr` are urgently needed by bioinformatics research community and will be a timely addition to the catalogue of sequence analysis tools. Further, the interoperability of `bedr` data structures with *BED* and *GRanges* *data.frame*/objects makes it an easy-

to-fit component in existing genomic pipelines. **bedr** is freely available as an open-source package through CRAN and lends itself for customized extensions needed for in-house sequencing-analysis pipelines as well as future bioinformatics protocols.

Availability and requirements

Project name: bedr

Project home page: <http://cran.r-project.org/web/packages/bedr>

Operating system(s): OSX, Linux/Unix

Programming language: R

Other requirements: BEDTools, BEDOPS

License: e.g. GNU GPL-2

Any restrictions to use by non-academics: None

Additional file

Additional file 1: Example workflow. (DOCX 30 kb)

Abbreviations

API: Application programming interface; BED: Browser extensible display; chr: Chromosome; COSMIC: Catalogue of somatic mutations in cancer; CRAN: The comprehensive R archive network; DAS: Distributed annotation system; GFF: General feature format; GTF: Gene transfer format; HUGO: Human Genome Organisation; VCF: Variant call format

Acknowledgements

We thank the Boutros lab for support and software testing.

Funding

This study was supported by the Ontario Institute for Cancer Research (PCB), the Ontario Ministry of Health and Long Term Care (OMOHLTC; FFL), the Dr. Mariano Elia Chair in Head & Neck Cancer Research, and philanthropic support from the Wharton family, Joe's Team, and Gordon Tozer. Views expressed do not necessarily reflect those of the OMOHLTC. PCB was supported by TFRI New Investigator and CIHR New Investigator Awards. This work was supported by Prostate Cancer Canada and is proudly funded by the Movember Foundation - Grant #RS2014-01. EL was supported by an Ontario Graduate Scholarship.

Availability of data and material

Software is available through CRAN: <http://cran.r-project.org/web/packages/bedr>. Data sharing not applicable to this article as no datasets were generated during the current study.

Authors' contributions

DW and PCB conceived the project. DW wrote the package with contributions and extensions from SH. SH, EL and CF performed testing and contributed to documentation and use-cases. PCB and FFL supervised the development. SH and PCB wrote the manuscript, which all authors read and approved.

Competing interests

The authors declare that they have no competing interests.

Consent for publication

No applicable.

Ethics approval and consent to participate

No applicable.

Author details

¹Informatics and Biocomputing Platform, Ontario Institute for Cancer Research, Toronto M5G 0A3, Canada. ²Departments of Radiation Oncology, Pharmacology & Toxicology, and Medical Biophysics, University of Toronto,

Toronto M5G 2M9, Canada. ³Ontario Cancer Institute and Campbell Family Institute for Cancer Research, Princess Margaret Hospital, University Health Network, Toronto M5G 2M9, Canada.

Received: 23 December 2015 Accepted: 30 September 2016

Published online: 15 December 2016

References

- Karolchik D, Hinrichs AS, Furey TS, Roskin KM, Sugnet CW, Haussler D, Kent WJ. The UCSC Table Browser data retrieval tool. *Nucleic Acids Res.* 2004;32:D493–96.
- Dowell RD, Jakerst RM, Day A, Eddy SR, Stein L. The distributed annotation system. *BMC Bioinformatics.* 2001;2:7.
- Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, Bravo HC, Davis S, Gatto L, Girke T, et al. Orchestrating high-throughput genomic analysis with Bioconductor. *Nat Methods.* 2015;12:115–21.
- Lawrence M, Huber W, Pages H, Aboyoun P, Carlson M, Gentleman R, Morgan MT, Carey VJ. Software for computing and annotating genomic ranges. *PLoS Comput Biol.* 2013;9:e1003118.
- Neph S, Kuehn MS, Reynolds AP, Haugen E, Thurman RE, Johnson AK, Rynes E, Maurano MT, Vierstra J, Thomas S, et al. BEDOPS: high-performance genomic feature operations. *Bioinformatics.* 2012;28:1919–20.
- Quinlan AR, Hall IM. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics.* 2010;26:841–2.
- Li H. Tabix: fast retrieval of sequence features from generic TAB-delimited files. *Bioinformatics.* 2011;27:718–9.
- Rosenbloom KR, Armstrong J, Barber GP, Casper J, Clawson H, Diekhans M, Dreszer TR, Fujita PA, Guruvadoo L, Haeussler M, et al. The UCSC Genome Browser database: 2015 update. *Nucleic Acids Res.* 2015;43:D670–81.
- Forbes SA, Beare D, Gunasekaran P, Leung K, Bindal N, Boutselakis H, Ding M, Bamford S, Cole C, Ward S, et al. COSMIC: exploring the world's knowledge of somatic mutations in human cancer. *Nucleic Acids Res.* 2015;43:D805–11.
- Gray KA, Yates B, Seal RL, Wright MW, Bruford EA. Genenames.org: the HGNC resources in 2015. *Nucleic Acids Res.* 2015;43:D1079–85.
- Chen H, Boutros PC. VennDiagram: a package for the generation of highly-customizable Venn and Euler diagrams in R. *BMC Bioinformatics.* 2011;12:35.
- Lex A, Gehlenborg N. Points of view: Sets and intersections. *Nat Methods.* 2014;11:779.
- Hahne F, Ivanek R. Visualizing genomic data using Gviz and Bioconductor. *Methods Mol Biol.* 2016;1418:335–51.

Submit your next manuscript to BioMed Central and we will help you at every step:

- We accept pre-submission inquiries
- Our selector tool helps you to find the most relevant journal
- We provide round the clock customer support
- Convenient online submission
- Thorough peer review
- Inclusion in PubMed and all major indexing services
- Maximum visibility for your research

Submit your manuscript at
www.biomedcentral.com/submit

