Bioinformatics, 38(13), 2022, 3327–3336
https://doi.org/10.1093/bioinformatics/btac327
Advance Access Publication Date: 16 May 2022
Original Paper

OXFORD

Genome analysis

Assessing and assuring interoperability of a genomics file format

Yi Nian Niu [1], **Eric G. Roberts** [1], **Danielle Denisko** [1,2] and **Michael M. Hoffman** [1,2,3,4,*]

[1]Princess Margaret Cancer Centre, University Health Network, Toronto, ON, M5G 1L7, Canada, [2]Department of Medical Biophysics, University of Toronto, Toronto, ON M5G 1L7, Canada, [3]Department of Computer Science, University of Toronto, Toronto, ON M5S 2E4, Canada and [4]Vector Institute for Artificial Intelligence, Toronto, ON M5G 1M1, Canada

*To whom correspondence should be addressed.
Associate Editor: Tobias Marschall

Received on January 7, 2022; revised on March 30, 2022; editorial decision on April 25, 2022; accepted on May 11, 2022

Abstract

Motivation: Bioinformatics software tools operate largely through the use of specialized genomics file formats. Often these formats lack formal specification, making it difficult or impossible for the creators of these tools to robustly test them for correct handling of input and output. This causes problems in interoperability between different tools that, at best, wastes time and frustrates users. At worst, interoperability issues could lead to undetected errors in scientific results.

Results: We developed a new verification system, Acidbio, which tests for correct behavior in bioinformatics software packages. We crafted tests to unify correct behavior when tools encounter various edge cases—potentially unexpected inputs that exemplify the limits of the format. To analyze the performance of existing software, we tested the input validation of 80 Bioconda packages that parsed the Browser Extensible Data (BED) format. We also used a fuzzing approach to automatically perform additional testing. Of 80 software packages examined, 75 achieved less than 70% correctness on our test suite. We categorized multiple root causes for the poor performance of different types of software. Fuzzing detected other errors that the manually designed test suite could not. We also created a badge system that developers can use to indicate more precisely which BED variants their software accepts and to advertise the software's performance on the test suite.

Availability and implementation: Acidbio is available at https://github.com/hoffmangroup/acidbio.

Contact: michael.hoffman@utoronto.ca

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

1.1 File format interoperability

For your latest research project, you have constructed a pipeline from multiple published bioinformatics tools. Each tool works well with the author's data, but you run into errors with your data. The author's data and your data have slight differences in file metadata and data formatting, which lead to the errors. As a result, you must spend time manually editing your data files and intermediate outputs to conform to each tool's expectations. Meanwhile, ensuring interoperability between software tools that parse the data file format could have prevented your frustration.

Scientific software developed by academics often suffers from software engineering deficiencies (Crouch *et al.*, 2013), which can lead to the scenario described above. Among these include problems with deployment (Mangul *et al.*, 2019), maintenance (Schultheiss, 2011), robustness (Taschuk and Wilson, 2017) and documentation (Karimzadeh and Hoffman, 2018). Software engineering flaws may hinder fulfilling the Findable, Accessible, Interoperable and Reusable (FAIR) principles for scientific data management (Wilkinson *et al.*, 2016)—especially the guidelines on interoperability and reusability. Software engineering flaws may also affect web services that parse bioinformatics file formats, which may have vulnerabilities to attacks such as malicious code injections in input files (Pauli, 2013).

One key difficulty arises from interoperability of specialized file formats used for scientific data. Often, creators specify such formats informally, or not at all, leaving users and developers to guess the details of critical components or edge cases. Rare standardization efforts such as those of the Global Alliance for Genomics and Health (GA4GH) (Rehm *et al.*, 2021) have developed a few formal specifications. These include the Sequence Alignment/Map (SAM),

BAM, CRAM and Variant Call Format (VCF) file formats (Global Alliance for Genomics and Health, 2022).

Interoperability issues can also arise from issues within the software. Developers can address some interoperability problems, however, through simple solutions such as checklists. For example, Bioconda (Grüning *et al.*, 2018) recipes require adequate tests and a stable source code uniform resource locator (URL) (Bioconda, 2022). Bioconductor (Gentleman *et al.*, 2004) also has guidelines for package submission regarding code style, performance and testing (Bioconductor, 2022). Simple checklists can greatly improve software quality, even for programmers and researchers that lack formal software engineering training.

Software testing recommendations and standard test suites can aid researchers and developers. Extensive test suites for common standards, such as TeX's trip tests (Knuth, 1984), or the Web Standards Project Acid test suite (Hickson, 2005) exercise independent implementations of common standards by focusing on edge cases. In a bioinformatics context, tools that parse VCF (Danecek *et al.*, 2011) can use simulated VCF files with known behavior to test software correctness (Yang *et al.*, 2017).

Here, we tackled the bioinformatics software engineering problem of file format interoperability, specifically focusing on the plain-text whitespace-delimited Browser Extensible Data (BED) format (Kent *et al.*, 2010). We chose to use the BED file format because of its simplicity and its popularity.

Many software tools have taken a liberal approach to accepting BED files. This seemingly increases the utility of these tools, but removes an incentive for BED producers to be meticulous about interoperability (Allman, 2011). Programmers may unwittingly create software that generates incorrect BED files if they only supply their output to downstream consumers with a liberal approach to validation. This results in technical debt, where problems lay undiscovered until after the developers complete the project, or years later, when it becomes much harder to fix.

At the inception of this work, the BED format did not have a comprehensive specification, but we considered such a specification a prerequisite for this work. Therefore, first, we developed a formal specification for the BED format, working with relevant stakeholders and soliciting public comments. We then shepherded the specification through the GA4GH standards process until it achieved formal approval. Second, we quantified the degree to which a wide variety of bioinformatics software varied in their processing of this file format. In particular, we tested bioinformatics software input validation, checking input data for correct formatting.

To facilitate this work, we created Acidbio (https://github.com/hoffmangroup/acidbio), a system for automated testing and certification of bioinformatics file format interoperability.

1.2 The BED file format

The BED format describes genomic intervals in plain text. Each BED file consists of a number of lines, each with 3–12 whitespace-delimited fields. The mandatory first three fields (chrom, chromStart and chromEnd) define an interval on a chromosome. The optional last nine fields provide additional information about the interval such as a name, score, strand and aesthetic features used by the University of California, Santa Cruz (UCSC) Genome Browser (Haeussler *et al.*, 2019). The optional fields have a required order—all fields preceding the last field used must contain values.

BED variants distinguish BED files based on its number of fields. BED*n* denotes a file with only the first *n* fields. For example, a BED4 file has the chrom, chromStart, chromEnd and name fields. BED3 to BED9, along with BED12, represent the 8 standard BED variants.

BED*n*+*m* denotes a file with the first *n* fields followed by *m* fields of custom-defined fields supplied by the user. The custom-defined fields can contain many types of plain-text data. BED*n*+*m* files act as custom BED files. Currently, no in-band information exists to supply information about a BED file's fields. A BED parser must infer the fields present in a BED file.

The file conversion tool bedToBigBed (Kent *et al.*, 2010), developed by the UCSC Genome Browser team, has served as the de facto file validation tool for the BED format (H.Clawson, personal communication). The BED format appears deceptively simple, and without careful consideration of the specification, a developer may miss unexpected flexibility or rigidity in some fields.

2 Materials and methods

2.1 The Acidbio test system

We developed the Acidbio test system, which automatically runs a number of bioinformatics tools on a test suite (Fig. 1). To determine an actual success or failure, we consider the exit status and outputs to standard output and error. A test case passes on a successful exit status and no error or warning messages printed.

We identified error and warning messages by manually running the tools. We had to identify these error and warning messages manually because some tools logged errors without returning a non-zero exit code or logged issues in the BED file through warnings instead of errors.

To provide Acidbio with details on how to run each tool, we created a YAML Ain't Markup Language (YAML) configuration file that stored each tool's command-line usage file (Fig. 2). The YAML file also stored the locations of the additional files needed to run each tool and each tool's Conda environment.

2.2 Tool discovery

To identify tools to test, we used Bioconda (Grüning *et al.*, 2018), a repository that contains thousands of bioinformatics software packages. Each package contains one or more tools. We only included Bioconda packages with tools that have a command-line interface, as opposed to add-on modules executed within another program, and use the BED format as input. This excluded the numerous R, Bioconductor (Gentleman *et al.*, 2004) and Perl packages that have no command-line interface.

For packages that contain multiple tools, we selected a smaller set of subtools to test. We systematically identified these packages by manually examining the documentation for over 1000 packages

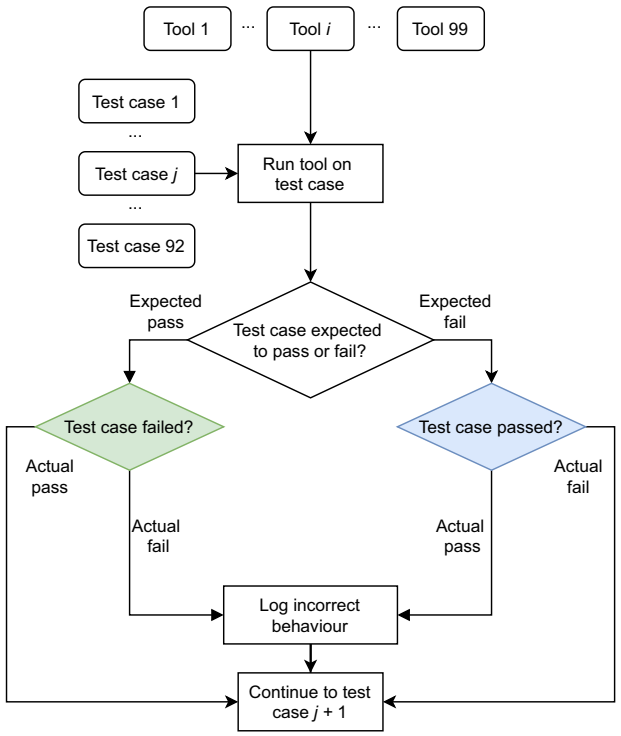

Fig. 1. Flowchart depicting how Acidbio evaluates tools on the test suite. Rounded rectangles: inputs; sharp rectangles: procedures; rhombuses: conditional branches. All 99 tools run on all 92 test cases. After the 92nd test case, Acidbio moves onto the next tool and runs it on the first test case

```
settings:
  file-locations:
    BAM: data/toy.bam
    SIZES: data/hg38.chrom.sizes
tools:
  - samtools:
    bedcov: samtools bedcov FILE BAM
  - ucsc:
    bedSort: bedSort FILE /dev/stdout
    bedClip: bedClip FILE SIZES /dev/stdout
    bedRemoveOverlap: bedRemoveOverlap FILE /dev/stdout
    bedToBigBed: bedToBigBed FILE SIZES TMPDIR/out.bb
conda-environment:
  samtools: base
  ucsc: base
```

Fig. 2. Excerpt from the Acidbio configuration file. The configuration file contains three sections. The 'settings' section lists the location of files or directories that Acidbio will insert into command-line execution. The 'tools' section contains the command-line invocations of the tested tools. In each invocation, Acidbio replaces 'FILE' with the location of the test BED file. The 'conda-environment' section lists each tool's Conda environment name

to determine whether they matched our criteria. We had to manually examine documentation because Bioconda has no structured metadata on each package's input file formats. This process yielded 80 packages, with 99 tools total.

Some tools use the BED format as the primary input file, such as a mandatory argument. Examples include bedtools (Quinlan and Hall, 2010) and high-throughput sequencing toolkits such as ngs-bits (Sturm *et al.*, 2018). These tools generally perform calculations using the intervals found in the BED file.

Other tools use the BED format as a secondary input file, such as an optional argument. Tools that use BED as a secondary input file generally use it to define genomic intervals of interest for data in another file format, such as SAM. In the tools we tested, 60 packages used the BED format as the primary input file, and 20 packages used the BED format as a secondary input file.

After collecting a list of all the possible packages that we could test, we then attempted to install each package and run the tools. We excluded packages that we could not install or could not run without error on any input files. We found no cases where a package contained both working and broken tools.

2.3 Test suite

We created a test suite that contains tests for each BED*n* format, covering various edge cases drawn from our BED specification. The test suite contains both expected success test cases (Supplementary Table S2) and expected fail test cases (Supplementary Table S3). Some tests include validating ranges for numeric fields, validating character sets for alphanumeric fields or data formatting for fields such as itemRgb or the block definitions.

We manually generated the test cases, designing them to make sense for all the tools tested. We used genomic intervals between positions 250 000 and 260 000 since one might find them in both chromosomes and non-chromosome scaffolds. Each test case varies based on the criteria tested. Some criteria only require a deviation in one field in one feature to generate a test case. For example, to test a score greater than 1000, only a single feature had a score greater than 1000. Other criteria required deviation in multiple features to generate a test case. For example, to test that the parser accepts strand '.', we set all features to strand '.'.

We built tests upon each other—we repeated a test case for all BED variants with additional fields added. As an example, a test case in BED5 testing a negative score gets repeated in testing the BED6 through BED12 variants.

For tools that use BED as a secondary file format, we collected test files for their non-BED primary file formats. For each of these file formats, we sourced an example file from the creators of the format or from a repository such as a FASTA for GRCh38/hg38 (Schneider *et al.*, 2017) from the UCSC Genome Browser (https://hgdownload.cse.ucsc.edu/goldenpath/hg38/bigZips/). We edited non-BED files to ensure that their ranges matched the BED test cases. We also validated the collected non-BED files with a file validator, when possible.

Since the new formal BED specification prohibits BED10 and BED11, we considered all BED10 and BED11 tests expected fail, even if the test case fell under expected success for other BED variants.

2.4 Fuzzing

We used a fuzzing approach (Miller *et al.*, 1990) to automatically generate test cases beyond our manually designed test suite (Fig. 3). We created an ANother Tool for Language Recognition 4 (ANTLR4) grammar (Parr *et al.*, 2014) to define the structure of the BED format and the possible values for each field. Then, we used a file generator that builds a file based on our grammar. We tested the tools using grammar-based fuzzing and grammarinator (Hodován *et al.*, 2018) as the file generator.

To introduce further variation into the BED file, we created an ANTLR4 meta-grammar that defines possible ANTLR4 BED grammars. The meta-grammar produces variation by allowing the BED grammar to vary on the structure or definition of fields. For example, the meta-grammar may produce a BED grammar that only allows tabs as the whitespace, or it may produce a BED grammar that allows both tabs and spaces. By varying the BED grammar produced, the user can test different combinations of field definitions and BED file structure that a single BED grammar cannot achieve.

3 Results

3.1 A new formal specification addresses ambiguities in the BED format

Despite existing for almost two decades, the BED format until recently lacked a formal specification similar to the SAM (Li *et al.*, 2009) or VCF (Danecek *et al.*, 2011) specification. The UCSC Genome Browser Data File Formats Frequently Asked Questions (https://genome.ucsc.edu/FAQ/FAQformat.html) specified some details, but lacked technical details that other formal specifications clearly define.

Through the GA4GH standards process (Rehm *et al.*, 2021), we established a specification of the BED format (https://samtools.github.io/hts-specs/BEDv1.pdf). The new specification defines each BED field and their possible numerical range or valid character patterns. It also provides semantics surrounding whitespace, sorting and default field values. The specification formalizes missing details and captures the existing use of the BED format, taking the input from relevant stakeholders into account. During the development of the specification, we solicited input from a number of stakeholders, including the UCSC Genome Browser team, the File Formats subgroup within the GA4GH Large Scale Genomics work stream (https://www.ga4gh.org/work_stream/large-scale-genomics/), and the public through GitHub comments (https://github.com/samtools/hts-specs/pull/570).

3.2 Most existing tools perform poorly on a BED test suite

To measure the ability of BED parsers to accept good input and reject bad input, we created an Acidbio test suite with 92 individual test cases. Specifically, we used the new specification to develop a test suite of expected pass and expected fail BED files. The expected pass test cases conform to our specification—for these cases, we expect tools to return a zero exit code and not output any error or warning messages. The expected fail test cases do not conform to our specification—for these cases, we expect tools to return a non-zero exit code or output an error or warning message. The test suite contains 92 tests, covering the definitions of fields and the structure of the BED file. The test suite also covers all BED variants from BED3 to BED12. The BED3 test cases represent the core of our test suite, as all BED files must have the first three fields.

The BED format does not contain in-band information on whether a file uses BED fields only or also has custom fields. A parser might assume that for BED files with 4–12 fields, all the

Fig. 3. Flowchart depicting grammar-based fuzzing. Rounded rectangles: files; sharp rectangles: generators. The BED meta-grammar describes different BED grammars to introduce further variation in the generated BED file

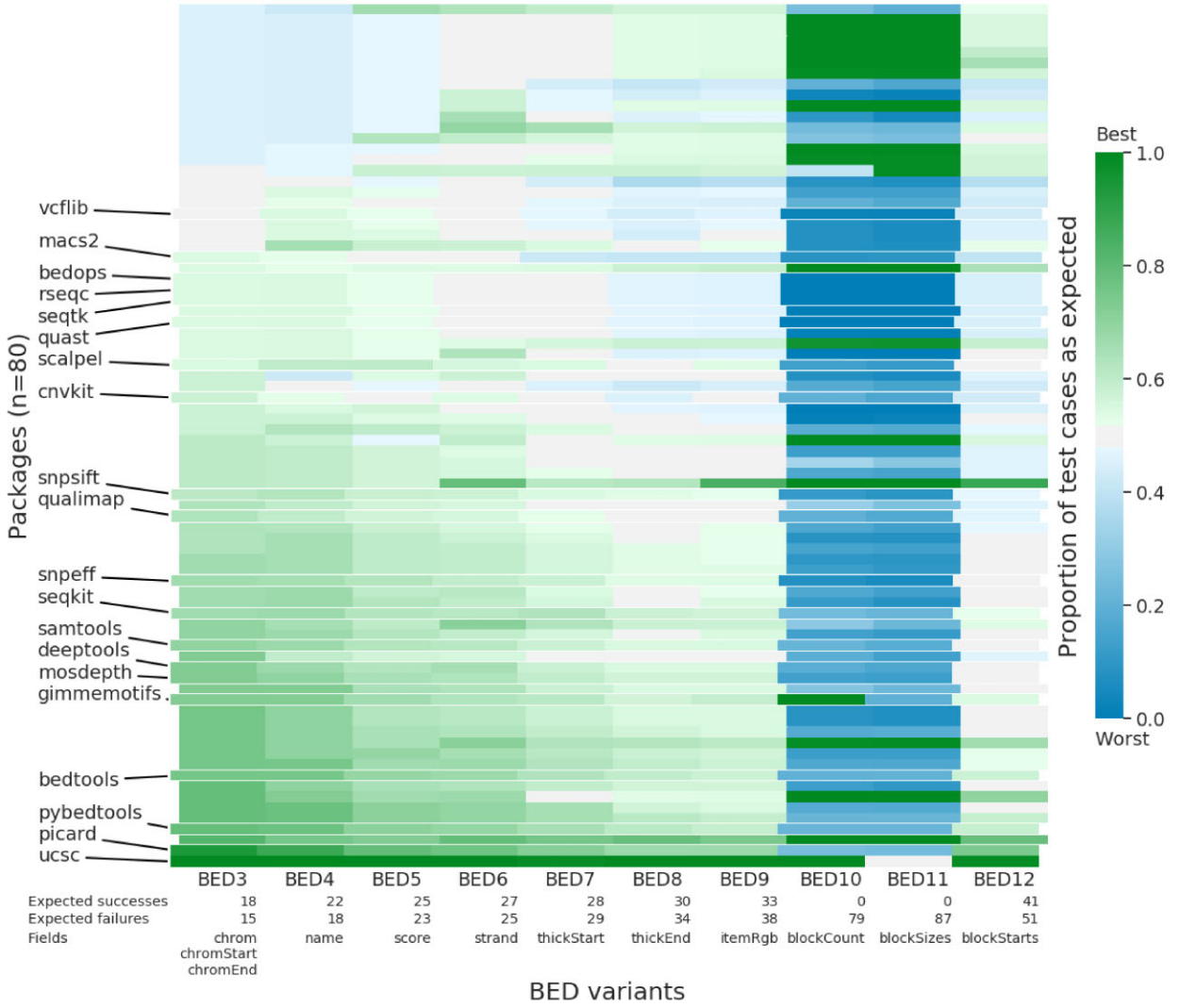

Fig. 4. Heatmap of performance of 80 Bioconda packages on 10 BED variants. Each cell shows the proportion of successful tests from the BED variant. Green cells: strong performance on the test suite; blue cells: poor performance. An expected success test case that succeeds or an expected fail test case that fails both represent a successful test. For packages with multiple tools, we display only results from the package's best-performing tool. Labeled, negative indented rows emphasize the 20 packages most downloaded from Bioconda. Rows sorted by ascending performance on the mandatory BED3 fields, then on performance of subsequent optional fields, ending with BED12 performance. The table below the heatmap lists the number of expected success and expected fail test cases for each BED variant. BED10 and BED11 have zero expected success test cases because the specification forbids BED10 and BED11 (A color version of this figure appears in the online version of this article.)

fields represent standard BED fields. In this case, the parser should validate the fields according to the file format rules.

Alternatively, a parser might treat fields 4 through 12 as custom data. A tool designed to handle arbitrary custom BED files may not validate the optional BED fields. This means the tool may not fail on the expected fail test cases. The expected success test cases, however, should all work even for non-specified custom data. Also, this flexibility does not apply to mandatory fields one through three, as their definition cannot change.

We examined behavior of tools, expecting strict validation of standard BED4 through BED12 files. This provides more informative results than permitting the whole range of behavior one might expect for custom data. Unexpected results in the optional fields indicate the need for better means for interchange of metadata on these fields.

Using our test suite, we assessed 80 Bioconda packages that support the BED format as input (Fig. 4). In some packages, we assessed multiple tools, making 99 tools in total. For each tool, we calculated its performance on each BED variant by taking the number of tests that behaved as expected divided by the number of tests for the BED variant. Of the 99 tools, only 26 achieved $\geq 70\%$ expected results for BED3 tests. Averaged for tests across all BED variants, 51 tools achieved $\geq 50\%$ expected results. We have deposited full results on Zenodo (https://doi.org/10.5281/zenodo.5784787). Beyond the possibility of expecting custom BED files, we attributed unexpected results to several causes described below.

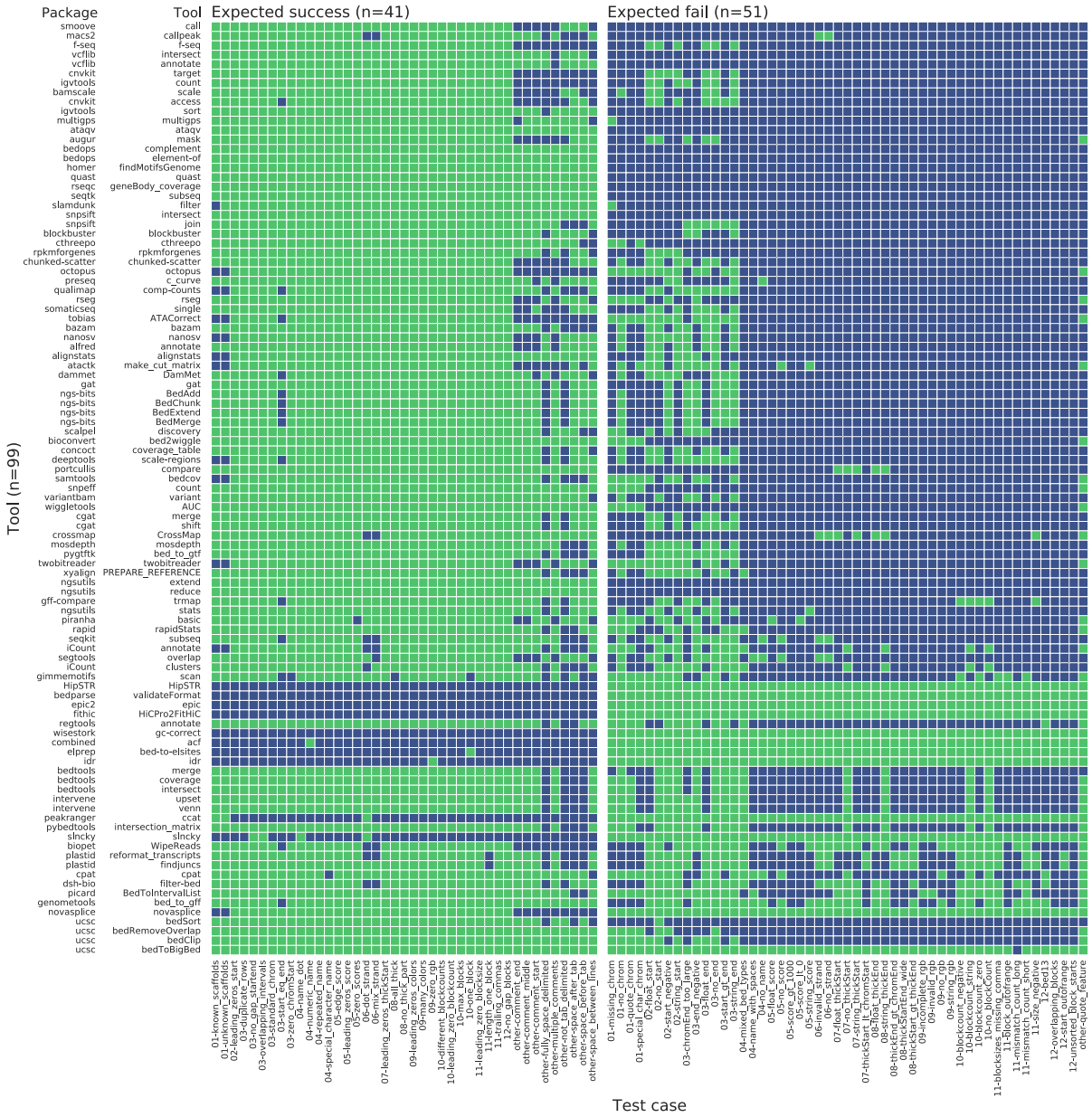

Fig. 5. Performance of 99 tools from 80 packages on 92 BED12 test cases (Alneberg *et al.*, 2014; Ay *et al.*, 2014; Bentsen *et al.*, 2020; Bioconvert Developers, 2017; Bollen *et al.*, 2019; Boyle *et al.*, 2008; Breese and Liu, 2013; Broad Institute, 2019; Buske *et al.*, 2011; Chen *et al.*, 2016; Cingolani *et al.*, 2012a; 2012b; Cooke *et al.*, 2021; Costanza *et al.*, 2019; Cotto *et al.*, 2021; Cretu Stancu *et al.*, 2017; T.Curk *et al.*, in preparation; Dale *et al.*, 2011; Daley and Smith, 2014; Dunn and Weissman, 2016; Fang *et al.*, 2015; 2016; Farek, 2017; Feng *et al.*, 2011; Garrison, 2012; Gremme *et al.*, 2013; Hanghøj *et al.*, 2019; Heger *et al.*, 2013; Heinz *et al.*, 2010; Hensly *et al.*, 2015; Herzeel *et al.*, 2015; 2019; Heuer, 2022; Huddleston *et al.*, 2021; Karunanithi *et al.*, 2019; Kaul, 2018; Kaul *et al.*, 2020; Kent *et al.*, 2002; Khan and Mathelier, 2017; Kodali, 2020; Langenberger *et al.*, 2009; Leonardi, 2019; Li, 2012; Li *et al.*, 2009; 2011; Lopez *et al.*, 2019; Mahony *et al.*, 2014; Mapleson *et al.*, 2018; Mikheenko *et al.*, 2018; Narzisi *et al.*, 2014; Neph *et al.*, 2012; Neumann *et al.*, 2019; Okonechnikov *et al.*, 2016; Orchard *et al.*, 2020; Pedersen, 2018; Pedersen *et al.*, 2012; Pedersen and Quinlan, 2018; Pertea and Pertea, 2020; Pongor *et al.*, 2020; Quinlan and Hall, 2010; Ramírez *et al.*, 2016; Ramsköld *et al.*, 2009; Rausch *et al.*, 2019; Robinson *et al.*, 2011; Sadedin and Oshlack, 2019; Schiller, 2013; Shen *et al.*, 2016; Sims *et al.*, 2014; Song and Smith, 2011; Stovner and Sætrom, 2019; Sturm *et al.*, 2018; Talevich *et al.*, 2016; Thorvaldsdóttir *et al.*, 2013; Uren *et al.*, 2012; van Heeringen and Veenstra, 2011; van't Hof *et al.*, 2017; Vorderman *et al.*, 2019; Wala *et al.*, 2016; Wang *et al.*, 2012; 2013; Webster *et al.*, 2019; Willems *et al.*, 2017; Xu *et al.*, 2010; Zerbino *et al.*, 2014; Zhang *et al.*, 2008; Zhao *et al.*, 2014). Green: the tool performed as expected; blue: the tool did not perform as expected. Rows sorted ascending by the number of test cases with an expected result. We grouped tools in the same package together as they tend to have similar results. For packages with multiple tools, we sorted the package using the best-performing tool. Within the same package, we sorted tools by ascending performance (A color version of this figure appears in the online version of this article.)

3.3 Existing tools parse BED files in different ways

All tools have distinct purposes, causing them to parse the BED format in different ways and focus on varying aspects of BED files. Different purposes mean some test cases may never arise in the expected usage of the tool. We have identified a few groups of tools that have similar behaviors, which cause poor performance on the test suite.

Tools that require a specific BED variant. Some tools require a specific number of fields in the input BED file. For example, slncky (Chen *et al.*, 2016) requires a BED12 file. This causes all BED3 to BED11 inputs to raise an error.

Tools that only validate a subset of BED fields. Many tools use the BED format only for interchange of genomic intervals in the first

three fields. Some of these tools will accept any BED*n* file and perform no validation after the first three fields. For example, many tools ignore fields that describe aesthetic features only for genomic browser display, such as thickStart, thickEnd and itemRgb. A tool such as bedtools (Quinlan and Hall, 2010) that mainly operates on genomic intervals would incorrectly succeed on an expected fail BED9 test case.

File converters. Some tools convert the BED format to a different file format, without performing any validation. Some file converters use a garbage-in-garbage-out approach, going from invalid input in BED format to invalid output in some other format. For example, bioconvert bed2wiggle (Bioconvert Developers, 2017) fails as expected on most expected fail test cases, but still produces output retaining the input file errors. Using a garbage-in-garbage-out approach may make debugging complex pipelines more difficult. Raising warnings during file conversion helps debugging, as the user can narrow down the source of the error to steps before file conversion.

Tools that use another library for BED parsing. Some tools call an external library to perform operations on BED files. If the main tool does not perform extra error checking of its own, it can only detect the same errors that the external library finds. For example, intervene (Khan and Mathelier, 2017) uses bedtools as a dependency, which results in their similar patterns of performance.

3.4 Ambiguous format specification makes uniform behavior more difficult

The previous absence of a formal specification for the BED format also influenced test performance. Inevitably, tools addressed edge cases heterogeneously when the lack of formal specification made the expected behavior non-obvious. Our formal specification and the behavior of the reference implementation bedToBigBed conflict with the expectations of tool developers in many ways.

Definition of whitespace. Many BED files use tabs to delimit fields. The BED format, however, also accepts spaces to delimit fields, if the fields themselves contain no spaces (H.Clawson, personal communication). Of the 99 tools examined, 60 reject space-delimited BED files allowed by the specification (Supplementary Table S1, 'other-fully_space_delimited.bed'). Also, the BED format permits blank lines, though 37 tools do not accept this (Supplementary Table S1, 'other-space_between_lines.bed').

Expanded definition of fields. The BED format requires strict limits for certain fields and some generators do not respect these limits. For example, the specification defines score as an integer value between 0 and 1000, inclusive. Some tools use the score as a *P*-value, which violates the integer definition. To allow tools to repurpose the nine optional fields, one can treat these tools as BED*n*+*m* parsers, with custom definitions for the remaining fields. Nonetheless, repurposing field names, such as score, with different definitions can confuse parsers that will misinterpret the data and use it incorrectly.

Conflict between our formal specification and bedToBigBed. We used the de facto file validator bedToBigBed to inform the design of our test suite. Without a formal specification, however, uncertainty surrounding specific edge cases arose when bedToBigBed disagreed with our understanding of correct behavior.

Our formal specification disagreed with bedToBigBed in three instances. First, bedToBigBed accepted a BED7 file with thickStart less than chromStart. Second, bedToBigBed accepted a BED12 file with the length of the blockSizes or blockStarts list greater than blockCount. Third, bedToBigBed accepted BED11 files while our specification disallowed BED11. The definitions of the above fields are in both Supplementary Tables S2 and S3.

3.5 Software engineering deficiencies lead to poor performance on the test suite

Beyond issues in differences in design between tools and the previous informal specification of the file format, we can also attribute poor testing performance to problems in software engineering. Given the previous underspecification of the BED format and the lack of test suites, however, we would recommend extreme caution before considering poor test performance as an indication of poor software quality for tools that existed before this article.

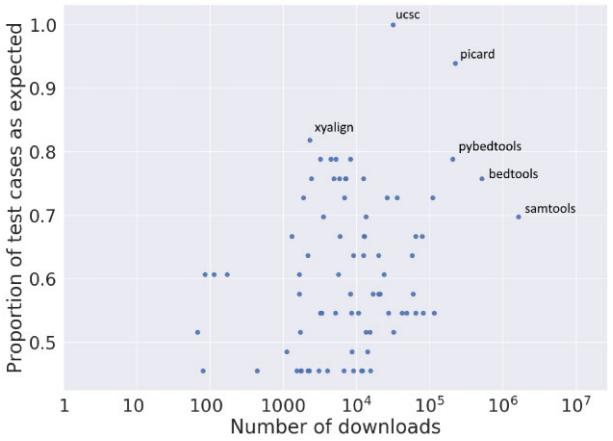

Fig. 6. Scatter plot of the number of downloads a package has on Bioconda against its performance on BED3 tests. Labeled points indicate the top four performing tools and the top four most downloaded tools. For packages with multiple tools, we display results from the best-performing tool

Silently accepting invalid input. Tools should alert users on input errors, allowing them to check whether they have made an error. In some cases, developers prefer to skip an invalid data point and continue. In this case, the tool should at least provide a warning message describing the skipped line. Otherwise, an error could slip past the user and affect their results. In our test suite, a warning message would count as an expected failure, improving the performance statistics for a tool that generates them.

Errors in BED file generators can easily slip past users. When a downstream tool raises an error on bad input, this reduces the time before someone discovers the problem with the upstream generator.

Insufficient testing. While some of our test cases cover formatting issues that can hinder interoperability, others represent 'can't happen' scenarios that, uncaught, pose logic bombs for a software tool. For example, all tools should reject negative start positions (Fig. 5), '02-negative-start.bed', but 48/99 tools accepted a test case that has negative starts. Given the limited resources and incentives to publish in academic software engineering, developers require a simpler way to ensure avoidance of obvious problems than manually developing test cases.

3.6 No relationship between package performance and downloads found

We observed little correlation between the number of downloads a package has compared to the package's performance on the test suite (Fig. 6). Many packages had a similar number of downloads. We attribute this to packages having specific purposes that make them useful for a few users. However, very highly downloaded packages such as bedtools (Quinlan and Hall, 2010) and the UCSC Genome Browser tool suite (Kent *et al.*, 2002) had better performance than other tools.

3.7 Automated fuzzing can detect errors that a manually designed test suite does not

Differential testing (McKeeman, 1998) using files generated from a grammar-based fuzzer (Godefroid *et al.*, 2008) can discover new errors not found by the test suite. A grammar-based fuzzer automatically generates files based on a defined structure of the file format.

We found one example of unexpected behavior in bedtools coverage (Quinlan and Hall, 2010) where coverage raised an error but bedToBigBed did not. Since bedtools coverage requires two input files, we generated two files using the fuzzer (Table 1) and validated them using bedToBigBed. On the generated files, bedtools

coverage exited with exit status 1 and error message 'Error: line number 1 of file 2.bed has 4 fields, but 0 were expected'. Our manually designed test suite did not catch this error—we only uncovered it due to the use of fuzzing.

3.8 BED badge indicates conformance with the BED format

We designed badges that developers can display in a tool's documentation to clearly indicate the file types used and indicate the tool's performance on the test suite (Fig. 7). The badges reassure users that the software underwent thorough testing. The availability of such badges encourages developers to perform input validation.

Acidbio includes steps to produce a BED badge. We recommend developers to display a BED badge if their software conforms to the BED formal specification.

4 Discussion

4.1 Use in software development

Acidbio can help researchers and programmers test their tools to improve the robustness and interoperability of their code. Acidbio can serve a similar function to the Web Standards Project Acid test suite (Hickson, 2005) designed to improve interoperability of web browsers. When the Web Standards Project created the Acid tests, many web browsers had poor compliance with existing web standards. Over time, browsers such as Opera (Gohring, 2006) and Internet Explorer (Schofield, 2007) began to achieve perfect performance on the Acid tests and interoperability improved. Similarly, we intend Acidbio to make it easier for developers to create bioinformatics software that more easily interoperates with other software.

To test new tools, developers need only create a short configuration YAML file to describe their tool's command line interface, and run the Acidbio test harness. From the test results, a programmer may identify edge cases they missed and fix them before distributing their software. Once fixed, the programmer can put a BED badge in a software's documentation to indicate that it interoperates with the BED format. Editors or reviewers of articles describing tools can use the test suite to verify the software's quality. Package repository managers can also use the test suite to verify the quality of submitted packages.

4.2 The utility of a formal specification

The interpretation of a standard can turn into a matter of opinion. While formalizing the standard with a specification can help improve interoperability, the only way to truly ensure agreement on expected behavior involves further formalization through a formal grammar or including test cases in the standard. A deterministic grammar or test suite removes potential for misunderstandings about standard conformance.

Table 1. Two files generated by a grammar-based fuzzer

File 1	File 2

`chr18 455914 533415 woG`	`chr12 632184 753365 Vx6`
	`#I`
	`#_`
	`#_`

4.3 Postel's law

Postel's law, 'be conservative in what you do, be liberal in what you accept from others' (Postel *et al.*, 1981), related initially to how software sends and accepts messages over the internet. Adherence to Postel's law helped the internet to succeed—leniency in accepting data without strict validation helped more organizations implement internet software (Bray, 2004).

The developers of the Extensible Markup Language (XML) format purposefully rejected Postel's law, deciding that malformed XML files would raise fatal errors (Bray, 2003). They did this because this approach encourages producers of the file format to strongly conform to the specification. A strict validation approach reduces opportunities for parsers to misunderstand input and prevents common errors from becoming accepted.

The lack of a strict validation approach for previous HyperText Markup Language (HTML) implementations led to a morass of incompatible and poorly described HTML file formats. This greatly increased the complexity of potential bugs in web browsers that could actually handle the existing base of web pages. Despite the existence of formal HTML specifications, web browsers had to create special 'quirks modes' to handle HTML files that did not satisfy these specifications (Olsson, 2014).

The history of HTML and XML should inform file validation behavior in bioinformatics software. While one may not want to raise fatal errors for each non-conforming file, BED parsers must at least provide warnings when encountering them. Users can easily ignore warnings, however, or miss them in a stream of irrelevant and voluminous diagnostic information. To ensure that users notice problems with file formats and that programmers fix upstream generators, parsers must take a strict validation 'warnings are errors' approach and refuse to parse invalid files.

4.4 Application to other bioinformatics file formats

Users and developers can apply the same methodology developed here to test other bioinformatics file formats for conformance. Establishing a common interface to parse a file format will improve interoperability of bioinformatics software and move closer to FAIR (Wilkinson *et al.*, 2016) goals. For binary file formats or software written in languages with weak memory safety, testing and interoperability become even more important.

Computational tools described in scholarly articles often undergo precious little testing. The existence of test systems such as Acidbio make it easy to test that a tool interoperates with other software well. We recommend that when such a test system exists, journal editors, reviewers and software repository managers ensure that the tool achieves good performance in the test suite prior to acceptance. For example, the European Variation Archive validates submitted VCF files against the VCF specification (Cezard *et al.*, 2022). After acceptance, repository managers can indicate which file formats the package uses as input and output to make searching for tools easier. Developers can also add badges similar to the BED

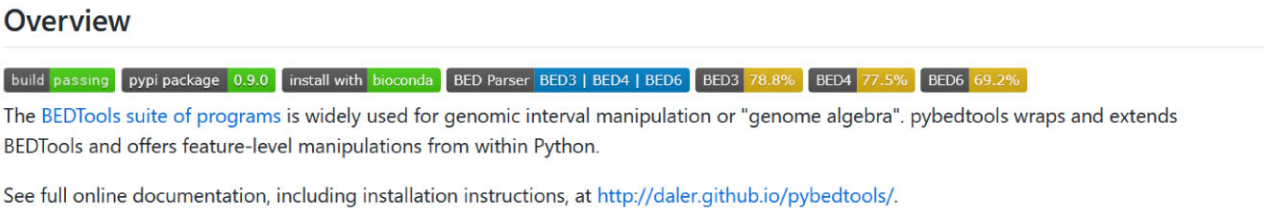

Fig. 7. Example of BED badges. BED badges allow developers to indicate the tool's support for the BED format and its conformance to the specification. The fourth badge, 'BED parser', displays the BED*n* formats the tool supports. The fifth through seventh badges displays the performance of the tool on the BED3, BED4 and BED6 variants. In this example, the tool passes 78.8% of BED3 tests, 77.5% of BED4 tests and 69.2% of BED6 tests

badge to indicate software's conformance to the relevant specification.

4.5 BED metadata

Tools parse BED files in the absence of in-band information embedded within the file. The lack of in-band information may lead to difficulties parsing BED files. For example, a tool cannot determine whether a BED file has custom fields without in-band information. This also makes testing tools properly more difficult. Without an idea of what BED variants a tool accepts, we cannot determine whether a test case suits the tool's intended use. With such metadata, tools can easily determine whether the input file has the fields it needs.

A header section at the beginning of a BED file can provide metadata to make parsing of BED files easier. The header can define the file's BED variant and specify information such as the genome assembly used. For custom BEDn+m files, the header can define the custom-defined fields, similar to the INFO lines in the VCF meta-information lines. Having a header would provide a direct method of supplying file metadata directly within the file, allowing parsers to easily read the BED file. Future versions of the GA4GH BED specification may add such metadata.

Future versions of the GA4GH BED specification may add metadata to provide essential information in-band. If this happens, an updated version of the test suite would need to incorporate the use of such metadata.

4.6 Limitations of the testing approach

Our testing approach applies the same BED files and secondary files to all the tools, except tools that use BAM input. Given the diversity of tools that use the BAM format, we could not find a single BAM file with data relevant to all tools. Instead, we used two different BAM files to avoid tools raising logical errors on our test cases.

More broadly, our testing applies the same criteria to all packages. The purposes of each package differ, but examining written documentation for all packages to apply specific tests for each presents an unfeasible challenge. Therefore, one should not regard poor performance on certain portions of the test suite alone as evidence of the quality of the software, which may otherwise remain fit for the purposes described. The previous underspecification of the BED format and the lack of test suites made consistent treatment of edge cases challenging for even very conscientious software developers. Nonetheless, now that a formal specification exists and the Acidbio test system testing for conformance to it easier, we recommend that future developers should ensure conformance with the specification to maximize interoperability with the rest of the BED ecosystem.

Our testing approach only considers whether a BED parser accepts valid input and rejects invalid input. It does not consider correctness of the output. Developers can validate output file format using a file validation tool. For BED files, one can use bedToBigBed (Kent *et al.*, 2010) for file validation, keeping in mind the edge cases discussed above where its behavior differs from the GA4GH BED specification. Testing for correctness of analyses represents a much more difficult problem that one cannot trivially address.

The fuzzing approach also has some limitations. The quality of the generated test cases relies on the file generator to cover a wide range of possible BED files. For a grammar-based fuzzing approach, the grammar would have to describe all possible variations in a file, which becomes difficult for more complex file formats. Another potential issue with file generation arises if the generator has too few methods to vary its output files, generating files that do not cover enough cases. Machine learning or other approaches that inform future file generation from past unexpected behavior can address this issue (Saavedra *et al.*, 2019).

Other fuzzing approaches, such as mutation-based fuzzing, may not work in a bioinformatics context. Mutation-based fuzzers randomly modify existing files by adding random or non-sense characters. These fuzzers would not create diverse BED files and the mutations would likely create invalid and meaningless BED files. A security-oriented fuzzer such as American Fuzzy Lop (Zalewski, 2018) can detect these vulnerabilities. Security-oriented fuzzers will produce test cases that can have nonsense data such as non-American Standard Code for Information Interchange (ASCII) characters, which tests the tool's ability to handle unexpected data.

Acknowledgements

The authors thank Carl Virtanen (0000-0002-2174-846X) and Zhibin Lu (0000-0001-6281-1413) at the University Health Network High-Performance Computing Centre and Bioinformatics Core for technical assistance, Michael Hicks (University of Maryland, College Park; 0000-0002-2759-9223) and Leonidas Lampropoulos (University of Maryland, College Park; 0000-0003-0269-9815) for helpful discussions, and W. James Kent and the UCSC Genome Browser team for creating the BED format.

Funding

This work was supported by the Natural Sciences and Engineering Research Council of Canada [RGPIN-2015-03948 to M.M.H.].

Conflict of Interest: none declared.

Data availability

Acidbio and the BED test suite are available in GitHub (https://github.com/hoffmangroup/acidbio) and deposited in Zenodo (https://doi.org/10.5281/zenodo.5784763). Results of each package on each test case and the scripts used to generate figures are available in Zenodo (https://doi.org/10.5281/zenodo.5784787).

References

Allman,E. (2011) The robustness principle reconsidered. *Commun. ACM*, **54**, 40–45.

Alneberg,J. *et al.* (2014) Binning metagenomic contigs by coverage and composition. *Nat. Methods*, **11**, 1144–1146.

Ay,F. *et al.* (2014) Statistical confidence estimation for Hi-C data reveals regulatory chromatin contacts. *Genome Res.*, **24**, 999–1011.

Bentsen,M. *et al.* (2020) ATAC-seq footprinting unravels kinetics of transcription factor binding during zygotic genome activation. *Nat. Commun.*, **11**, 4267.

Bioconda. (2022) *Guidelines for Bioconda Recipes*. https://bioconda.github.io/contributor/guidelines.html (22 March 2022, date last accessed).

Bioconductor. (2022) *Bioconductor – Package Submission*. https://www.bioconductor.org/developers/package-submission/ (22 March 2022, date last accessed).

Bioconvert Developers. (2017) *Bioconvert*. https://bioconvert.readthedocs.io/en/master/index.html (22 March 2022, date last accessed).

Bollen,S. *et al.* (2019) *sndrtj/wisestork: Version 0.1.0*. https://doi.org/10.5281/zenodo.3245885 (22 March 2022, date last accessed).

Boyle,A.P. *et al.* (2008) F-seq: a feature density estimator for high-throughput sequence tags. *Bioinformatics*, **24**, 2537–2538.

Bray,T. (2003) *Dracon and Postel*. https://www.tbray.org/ongoing/When/200x/2003/08/19/Draconianism (22 March 2022, date last accessed).

Bray,T. (2004) *On Postel, Again*. https://www.tbray.org/ongoing/When/200x/2004/01/11/PostelPilgrim (22 March 2022, date last accessed).

Breese,M.R. and Liu,Y. (2013) NGSUtils: a software suite for analyzing and manipulating next-generation sequencing datasets. *Bioinformatics*, **29**, 494–496.

Broad Institute (2019) *Picard Toolkit*. https://broadinstitute.github.io/picard/ (22 March 2022, date last accessed).

Buske,O.J. *et al.* (2011) Exploratory analysis of genomic segmentations with Segtools. *BMC Bioinformatics*, **12**, 415.

Cezard,T. *et al.* (2022) The European Variation Archive: a FAIR resource of genomic variation for all species. *Nucleic Acids Res.*, **50**, D1216–D1220.

Chen,J. *et al.* (2016) Evolutionary analysis across mammals reveals distinct classes of long non-coding RNAs. *Genome Biol.*, **17**, 19.

Cingolani,P. *et al.* (2012a) A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff. *Fly (Austin)*, **6**, 80–92.

Cingolani,P. *et al.* (2012b) Using *Drosophila melanogaster* as a model for genotoxic chemical mutational studies with a new program, SnpSift. *Front. Genet.*, **3**, 35.
Cooke,D.P. *et al.* (2021) A unified haplotype-based method for accurate and comprehensive variant calling. *Nat. Biotechnol.*, **39**, 885–892.
Costanza,P. *et al.* (2019) A comparison of three programming languages for a full-fledged next-generation sequencing tool. *BMC Bioinformatics*, **20**, 301.
Cotto,K.C. *et al.* (2021) RegTools: integrated analysis of genomic and transcriptomic data for the discovery of splicing variants in cancer. bioRxiv, 436634.
Cretu Stancu,M. *et al.* (2017) Mapping and phasing of structural variation in patient genomes using nanopore sequencing. *Nat. Commun.*, **8**, 1326.
Crouch,S. *et al.* (2013) The Software Sustainability Institute: changing research software attitudes and practices. *Comput. Sci. Eng.*, **15**, 74–80.
Dale,R.K. *et al.* (2011) Pybedtools: a flexible python library for manipulating genomic datasets and annotations. *Bioinformatics*, **27**, 3423–3424.
Daley,T. and Smith,A.D. (2014) Modeling genome coverage in single-cell sequencing. *Bioinformatics*, **30**, 3159–3165.
Danecek,P. *et al.* (2011) The variant call format and VCFtools. *Bioinformatics*, **27**, 2156–2158.
Dunn,J.G. and Weissman,J.S. (2016) Plastid: nucleotide-resolution analysis of next-generation sequencing and genomics data. *BMC Genomics*, **17**, 958.
Fang,L.T. *et al.* (2015) An ensemble approach to accurately detect somatic mutations using SomaticSeq. *Genome Biol.*, **16**, 197.
Fang,H. *et al.* (2016) Indel variant analysis of short-read sequencing data with Scalpel. *Nat. Protoc.*, **11**, 2529–2548.
Farek,J. (2017) *AlignStats*. https://github.com/jfarek/alignstats (22 March 2022, date last accessed).
Feng,X. *et al.* (2011) PeakRanger: a cloud-enabled peak caller for ChIP-seq data. *BMC Bioinformatics*, **12**, 139.
Garrison,E. (2012) *Vcflib: A C++ Library for Parsing and Manipulating VCF Files*. https://github.com/ekg/vcflib (22 March 2022, date last accessed).
Gentleman,R.C. *et al.* (2004) Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol.*, **5**, R80.
Global Alliance for Genomics and Health. (2022) *Genomic Data Toolkit*. https://www.ga4gh.org/genomic-data-toolkit/ (22 March 2022, date last accessed).
Godefroid,P. *et al.* (2008) Grammar-based whitebox fuzzing. In: *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '08*, Association for Computing Machinery, New York, NY, USA. pp. 206–215.
Gohring,N. (2006) *Acid Test May Prove New Browsers are Tough Sell*. https://www.networkworld.com/article/2309699/acid-test-may-prove-new-browsers-are-tough-sell.html (22 March 2022, date last accessed).
Gremme,G. *et al.* (2013) GenomeTools: a comprehensive software library for efficient processing of structured genome annotations. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **10**, 645–656.
Grüning,B. *et al.*; Bioconda Team. (2018) Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nat. Methods*, **15**, 475–476.
Haeussler,M. *et al.* (2019) The UCSC Genome Browser database: 2019 update. *Nucleic Acids Res.*, **47**, D853–D858.
Hanghøj,K. *et al.* (2019) DamMet: ancient methylome mapping accounting for errors, true variants, and post-mortem DNA damage. *GigaScience*, **8**, giz025.
Heger,A. *et al.* (2013) GAT: a simulation framework for testing the association of genomic intervals. *Bioinformatics*, **29**, 2046–2048.
Heinz,S. *et al.* (2010) Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities. *Mol. Cell.*, **38**, 576–589.
Hensly,J. *et al.* (2015) *ATACTK: A Toolkit for ATAC-Seq Data*. https://atactk.readthedocs.io/en/latest/index.html (22 March 2022, date last accessed).
Herzeel,C. *et al.* (2015) elPrep: high-performance preparation of sequence alignment/map files for variant calling. *PLoS One*, **10**, e0132868.
Herzeel,C. *et al.* (2019) elPrep 4: a multithreaded framework for sequence analysis. *PLoS One*, **14**, e0209523.
Heuer,M. (2022) *dishevelled-bio*. https://github.com/heuermh/dishevelled-bio (22 March 2022, date last accessed).
Hickson,I. (2005) *Acid2*. https://www.webstandards.org/files/acid2/test.html (22 March 2022, date last accessed).
Hodován,R. *et al.* (2018) Grammarinator: a grammar-based open source fuzzer. In: *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, *Lake Buena Vista, FL, USA*. pp. 45–48.
Huddleston,J. *et al.* (2021) Augur: a bioinformatics toolkit for phylogenetic analyses of human pathogens. *JOSS*, **6**, 2906.
Karimzadeh,M. and Hoffman,M.M. (2018) Top considerations for creating bioinformatics software documentation. *Brief. Bioinform.*, **19**, 693–699.
Karunanithi,S. *et al.* (2019) Automated analysis of small RNA datasets with RAPID. *PeerJ*, **7**, e6710.
Kaul,A. (2018) *Novasplice*. https://aryakaul.github.io/novasplice/ (22 March 2022, date last accessed).
Kaul,A. *et al.* (2020) Identifying statistically significant chromatin contacts from Hi-C data with FitHiC2. *Nat. Protoc.*, **15**, 991–1012.
Kent,W.J. *et al.* (2002) The human genome browser at UCSC. *Genome Res.*, **12**, 996–1006.
Kent,W.J. *et al.* (2010) BigWig and BigBed: enabling browsing of large distributed datasets. *Bioinformatics*, **26**, 2204–2207.
Khan,A. and Mathelier,A. (2017) Intervene: a tool for intersection and visualization of multiple gene or genomic region sets. *BMC Bioinformatics*, **18**, 287.
Knuth,D.E. (1984) A torture test for TeX. *Technical report*, Department of Computer Science, Stanford University.
Kodali,V. (2020) *cthreepo*. https://github.com/vkkodali/cthreepo (22 March 2022, date last accessed).
Langenberger,D. *et al.* (2009) Evidence for human microRNA-offset RNAs in small RNA sequencing data. *Bioinformatics*, **25**, 2298–2301.
Leonardi,T. (2019) Bedparse: feature extraction from BED files. *JOSS*, **4**, 1228.
Li,H. (2012) *Seqtk*. https://github.com/lh3/seqtk (22 March 2022, date last accessed).
Li,H. *et al.* (2009) The sequence alignment/map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.
Li,Q. *et al.* (2011) Measuring reproducibility of high-throughput experiments. *Ann. Appl. Stat.*, **5**, 1752–1779.
Lopez,F. *et al.* (2019) Explore, edit and leverage genomic annotations using Python GTF toolkit. *Bioinformatics*, **35**, 3487–3488.
Mahony,S. *et al.* (2014) An integrated model of multiple-condition ChIP-seq data reveals predeterminants of Cdx2 binding. *PLoS Comput. Biol.*, **10**, e1003501.
Mangul,S. *et al.* (2019) Challenges and recommendations to improve the installability and archival stability of omics computational tools. *PLoS Biol.*, **17**, e3000333.
Mapleson,D. *et al.* (2018) Efficient and accurate detection of splice junctions from RNA-seq with portcullis. *GigaScience*, **7**, giy131.
McKeeman,W.M. (1998) Differential testing for software. *Digit. Tech. J.*, **10**, 100–107.
Mikheenko,A. *et al.* (2018) Versatile genome assembly evaluation with QUAST-LG. *Bioinformatics*, **34**, i142–i150.
Miller,B.P. *et al.* (1990) An empirical study of the reliability of UNIX utilities. *Commun. ACM*, **33**, 32–44.
Narzisi,G. *et al.* (2014) Accurate de novo and transmitted indel detection in exome-capture data using microassembly. *Nat. Methods*, **11**, 1033–1036.
Neph,S. *et al.* (2012) BEDOPS: high-performance genomic feature operations. *Bioinformatics*, **28**, 1919–1920.
Neumann,T. *et al.* (2019) Quantification of experimentally induced nucleotide conversions in high-throughput sequencing datasets. *BMC Bioinformatics*, **20**, 258.
Okonechnikov,K. *et al.* (2016) Qualimap 2: advanced multi-sample quality control for high-throughput sequencing data. *Bioinformatics*, **32**, 292–294.
Olsson,M. (2014) *CSS Quick Syntax Reference Guide*. Apress, Berkeley, CA, USA.
Orchard,P. *et al.* (2020) Quantification, dynamic visualization, and validation of bias in ATAC-seq data with ataqv. *Cell Syst.*, **10**, 298–306.
Parr,T. *et al.* (2014) Adaptive LL (*) parsing: the power of dynamic analysis. *ACM SIGPLAN Not.*, **49**, 579–598.
Pauli,J. (2013) *The Basics of Web Hacking: Tools and Techniques to Attack the Web*. Elsevier, Waltham, MA, USA.
Pedersen,B.S. (2018) *Smoove*. https://github.com/brentp/smoove (22 March 2022, date last accessed).
Pedersen,B.S. and Quinlan,A.R. (2018) Mosdepth: quick coverage calculation for genomes and exomes. *Bioinformatics*, **34**, 867–868.
Pedersen,B.S. *et al.* (2012) Comb-p: software for combining, analyzing, grouping and correcting spatially correlated p-values. *Bioinformatics*, **28**, 2986–2988.
Pertea,G. and Pertea,M. (2020) GFF utilities: GffRead and GffCompare. *F1000Research*, **9**, 304.
Pongor,L.S. *et al.* (2020) BAMscale: quantification of DNA sequencing peaks and generation of scaled coverage tracks. *Epigenet. Chromatin*, **13**, 21.

Postel,J. *et al.* (1981) *Transmission Control Protocol, Request For Comments 793*. https://datatracker.ietf.org/doc/html/rfc793 (22 March 2022, date last accessed).

Quinlan,A.R. and Hall,I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**, 841–842.

Ramírez,F. *et al.* (2016) deepTools2: a next generation web server for deep-sequencing data analysis. *Nucleic Acids Res.*, **44**, W160–W165.

Ramsköld,D. *et al.* (2009) An abundance of ubiquitously expressed genes revealed by tissue transcriptome sequence data. *PLoS Comput. Biol.*, **5**, e1000598.

Rausch,T. *et al.* (2019) Alfred: interactive multi-sample BAM alignment statistics, feature counting and feature annotation for long- and short-read sequencing. *Bioinformatics*, **35**, 2489–2491.

Rehm,H.L. *et al.* (2021) GA4GH: international policies and standards for data sharing across genomic research and healthcare. *Cell Genomics*, **1**, 100029.

Robinson,J.T. *et al.* (2011) Integrative genomics viewer. *Nat. Biotechnol.*, **29**, 24–26.

Saavedra,G.J. *et al.* (2019) A review of machine learning applications in fuzzing. *arXiv*, 1906:11133.

Sadedin,S.P. and Oshlack,A. (2019) Bazam: a rapid method for read extraction and realignment of high-throughput sequencing data. *Genome Biol.*, **20**, 78.

Schiller,B.J. (2013) Data biology: a quantitative exploration of gene regulation and underlying mechanisms. PhD Thesis, University of California, San Francisco.

Schneider,V.A. *et al.* (2017) Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome Res.*, **27**, 849–864.

Schofield,J. (2007) *Internet Explorer 8 Passes Acid2 Test*. https://www.theguardian.com/technology/blog/2007/dec/21/internetexplorer8passesaci (22 March 2022, date last accessed).

Schultheiss,S.J. (2011) Ten simple rules for providing a scientific web resource. *PLoS Comput. Biol.*, **7**, e1001126.

Shen,W. *et al.* (2016) SeqKit: a cross-platform and ultrafast toolkit for FASTA/Q file manipulation. *PLoS One*, **11**, e0163962.

Sims,D. *et al.* (2014) CGAT: computational genomics analysis toolkit. *Bioinformatics*, **30**, 1290–1291.

Song,Q. and Smith,A.D. (2011) Identifying dispersed epigenomic domains from ChIP-seq data. *Bioinformatics*, **27**, 870–871.

Stovner,E.B. and Sætrom,P. (2019) epic2 efficiently finds diffuse domains in ChIP-seq data. *Bioinformatics*, **35**, 4392–4393.

Sturm,M. *et al.* (2018) ngs-bits short-read sequencing tools for diagnostics. In: *European Conference on Computational Biology, Athens, Greece*.

Talevich,E. *et al.* (2016) CNVkit: genome-wide copy number detection and visualization from targeted DNA sequencing. *PLoS Comput. Biol.*, **12**, e1004873.

Taschuk,M. and Wilson,G. (2017) Ten simple rules for making research software more robust. *PLoS Comput. Biol.*, **13**, e1005412.

Thorvaldsdóttir,H. *et al.* (2013) Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Brief. Bioinform.*, **14**, 178–192.

Uren,P.J. *et al.* (2012) Site identification in high-throughput RNA-protein interaction data. *Bioinformatics*, **28**, 3013–3020.

van Heeringen,S.J. and Veenstra,G.J.C. (2011) GimmeMotifs: a de novo motif prediction pipeline for ChIP-sequencing experiments. *Bioinformatics*, **27**, 270–271.

Van't Hof,P. *et al.* (2017) Biopet: towards scalable, maintainable, user-friendly, robust and flexible NGS data analysis pipelines. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Madrid, Spain*. pp. 823–829.

Vorderman,R. *et al.* (2019) *Chunked-Scatter*. https://github.com/biowdl/chunked-scatter (22 March 2022, date last accessed).

Wala,J. *et al.* (2016) VariantBam: filtering and profiling of next-generational sequencing data using region-specific rules. *Bioinformatics*, **32**, 2029–2031.

Wang,L. *et al.* (2012) RSeQC: quality control of RNA-seq experiments. *Bioinformatics*, **28**, 2184–2185.

Wang,L. *et al.* (2013) CPAT: coding-potential assessment tool using an alignment-free logistic regression model. *Nucleic Acids Res.*, **41**, e74.

Webster,T.H. *et al.* (2019) Identifying, understanding, and correcting technical biases on the sex chromosomes in next-generation sequencing data. *GigaScience*, **8**, giz074.

Wilkinson,M.D. *et al.* (2016) The FAIR guiding principles for scientific data management and stewardship. *Sci. Data*, **3**, 160018.

Willems,T. *et al.* (2017) Genome-wide profiling of heritable and de novo STR variations. *Nat. Methods*, **14**, 590–592.

Xu,H. *et al.* (2010) A signal-noise model for significance analysis of ChIP-seq with negative control. *Bioinformatics*, **26**, 1199–1204.

Yang,A. *et al.* (2017) Scalability and validation of big data bioinformatics software. *Comput. Struct. Biotechnol. J.*, **15**, 379–386.

Zalewski,M. (2018) *American Fuzzy Lop (2.52b)*. https://lcamtuf.coredump.cx/afl/ (22 March2022, date last accessed).

Zerbino,D.R. *et al.* (2014) WiggleTools: parallel processing of large collections of genome-wide datasets for visualization and statistical analysis. *Bioinformatics*, **30**, 1008–1009.

Zhang,Y. *et al.* (2008) Model-based analysis of ChIP-Seq (MACS). *Genome Biol.*, **9**, R137.

Zhao,H. *et al.* (2014) CrossMap: a versatile tool for coordinate conversion between genome assemblies. *Bioinformatics*, **30**, 1006–1007.