Bioinformatics, 38(10), 2022, 2943–2945
https://doi.org/10.1093/bioinformatics/btac166
Advance Access Publication Date: 21 March 2022
Applications Note

Gene expression

Analysing high-throughput sequencing data in Python with HTSeq 2.0

Givanna H. Putri [1,2], Simon Anders [3], Paul Theodor Pyl [4], John E. Pimanda [1,2,5,6] and Fabio Zanini [1,2,7,*]

[1]School of Clinical Medicine, University of New South Wales, Sydney, NSW 2033, Australia, [2]Adult Cancer Program, Lowy Cancer Research Centre, University of New South Wales, Sydney, NSW 2033, Australia, [3]Bioquant Center, University of Heidelberg, 69120 Heidelberg, Germany, [4]Division of Surgery, Oncology and Pathology, Department of Clinical Sciences Lund, Faculty of Medicine, Lund University, Lund, Sweden, [5]Department of Pathology, School of Medical Sciences, University of New South Wales, Sydney, NSW 2052, Australia, [6]Department of Haematology, The Prince of Wales Hospital, Sydney, NSW 2031, Australia and [7]Cellular Genomics Futures Institute, University of New South Wales, Sydney, NSW 2033, Australia

*To whom correspondence should be addressed.

Associate Editor: Valentina Boeva

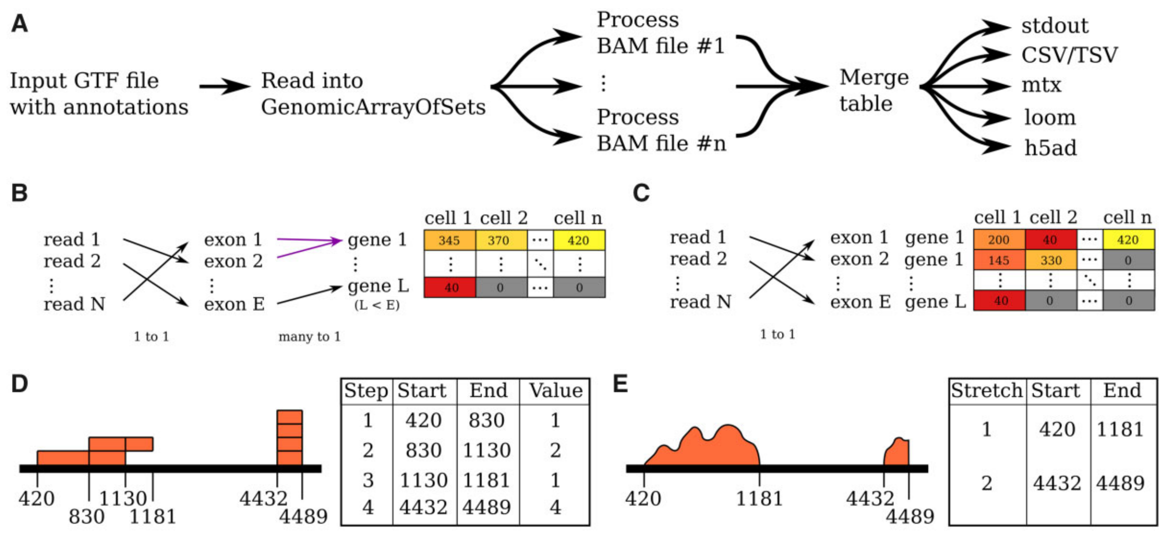
Received on November 17, 2021; revised on February 4, 2022; editorial decision on March 14, 2022; accepted on March 17, 2022

Abstract

Summary: HTSeq 2.0 provides a more extensive application programming interface including a new representation for sparse genomic data, enhancements for htseq-count to suit single-cell omics, a new script for data using cell and molecular barcodes, improved documentation, testing and deployment, bug fixes and Python 3 support.
Availability and implementation: HTSeq 2.0 is released as an open-source software under the GNU General Public License and is available from the Python Package Index at https://pypi.python.org/pypi/HTSeq. The source code is available on Github at https://github.com/htseq/htseq.
Contact: fabio.zanini@unsw.edu.au
Supplementary information: Supplementary data are available at *Bioinformatics* online.

Single-cell omics have exploded in popularity over the last few years, spearheaded by single cell transcriptomics. While commercial software solutions from manufacturers such as 10X Genomics and BD Biosciences provide standardized pipelines (e.g. *cellranger*) for analysing single-cell omics data, numerous experimental approaches rely on open source software to align reads and subsequently to quantify biological phenomena such as gene expression, chromatin accessibility, transcription factor binding affinities, and 3D chromatin conformation. *HTSeq* (Anders *et al.*, 2015) was initially developed as a general purpose tool to analyse high-throughput sequencing data in Python. In parallel, the *htseq-count* script was designed to count the number of reads or read pairs attributable to distinct genes in bulk RNA-Seq experiments. At that time, single-cell approaches were limited to specialized biotechnology laboratories. In this application note, we report the development of *HTSeq 2.0*, which improves the general-purpose application programming interface (API) and specifically *htseq-count* to encompass diverse omics analyses, including single-cell RNA sequencing (scRNA-Seq).

First, we have improved *htseq-count*, a popular script used to quantify gene expression in bulk and scRNA-Seq experiments (Fig. 1A–C). Multiple BAM files can now be processed with a single call of the script, which results in a counts table with either each row or column representing the counts from a separate BAM file. This is not only convenient but also faster because genomic features are loaded only once from the Gene transfer format (GTF) file, which can take as long as processing the reads for a typical plate-based single-cell experiment (Supplementary Fig. S1A and B). If multiple cores are available on the machine, *htseq-count* is now able to parallelize the quantification by allocating distinct input BAM files to each core (Fig. 1A, Supplementary Fig. S1A and B). The script also supports more output formats: compressed sparse matrices via *scipy* (Virtanen *et al.*, 2020), *mtx* files in the style of *cellranger*, h5-like file formats such as *h5ad* (Wolf *et al.*, 2018), and *loom* (http://loompy.org) (Fig. 1A). These output formats make it easier for users to import the counts table into downstream analysis libraries, especially single-cell ones such as *scanpy* (Wolf *et al.*, 2018) and *singlet* (https://github.com/iosonofabio/singlet). We also added support for storing additional metadata for each genomic feature. This has two clear applications: (i) Tracking additional gene information such as chromosome or aliases, which is useful for downstream analyses (e.g. for excluding sex chromosomes), and (ii) Collecting disaggregated exon-level counts, which provides a simple yet powerful approach to quantifying differential isoform expression (Fig. 1B). To encourage users to customize their analysis pipeline, we also restructured the key steps of *htseq-count* into well-documented functions and added a tutorial that explains the feature counting step by



Step	Start	End	Value
1	420	830	1
2	830	1130	2
3	1130	1181	1
4	4432	4489	4

Stretch	Start	End
1	420	1181
2	4432	4489

Fig. 1. Major *HTSeq 2.0* improvements. (**A–C**) Improvements to *htseq-count*. (A) Parallel processing on multicore architectures enables faster processing of single-cell data, where each cell is represented by a BAM file [typical for Smart-seq2 (Picelli et al. 2013) and viscRNA-Seq (Zanini *et al.*, 2018)]. Note the new output formats available in *HTSeq 2.0*. (B) Conventional gene–cell matrix, which collapses reads that align to distinct exons of the same gene into a single gene count. (C) Additional attributes enable quantification at the exon level while retaining information on which gene each exon belongs to. (**D, E**) Sparse data representations in *HTSeq 2.0*. (D) StepVector represents piecewise-constant sparse genomic data. (E) StretchVector represents sparse islands of genomic data

step. In addition, through a new script called *htseq-count-barcodes*, we support quantification of features in data multiplexed via cell barcodes and unique molecular identifiers (UMIs). Among other applications, the new script enables custom re-analysis of BAM files produced by *cellranger* using different parameters. Pearson correlation between *cellranger* and *htseq-count-barcodes* with default parameters is 0.985, with uniformly high correlation across cells (Supplementary Fig. S1C).

One of the key data structures in *HTSeq* is *StepVector*, an efficient sparse representation for piecewise-constant values on a 1D discrete space (typically a chromosome) (Fig. 1D). As an example, it can be used to store overlaps between gene bodies, critical for removing ambiguities in downstream gene expression analyses. However, genomic data is sometimes characterized by a distinct type of sparsity whereby the data appears as dense 'islands of knowledge' in a sea of missing data. This type of sparsity is apparent in the read coverage produced by amplicon sequencing or Chromatin Immunoprecipitation Sequencing (ChIP-Seq) where most of the genome is uncovered, but non-zero rapidly fluctuating coverage, down to a single nucleotide resolution (e.g. due to single nucleotide polymorphisms), are present only around specific kilobase-long stretches. To represent this type of sparsity efficiently, we created a new data structure called *StretchVector*. At its core, a *StretchVector* is a collection of stretches implemented via dense numpy arrays (Harris *et al.*, 2020), each with associated start-end coordinates (Fig. 1E). Each stretch represents an island of data, while the rest of the genome is not stored. We implemented functions for stretch extension, trimming, resetting, shifting, views or slices, copy and conversion to and from monolithic arrays for simple data ingestion/extraction. Separately from *StretchVector*, we also improved the support for custom ChIP-Seq and chromatin conformation capture (Hi-C) analyses by adding parsers for bedGraph and BigWig files via *pyBigWig* (Ryan *et al.*, 2021) and by writing new dedicated tutorials.

Finally, we improved the API of *HTSeq* as a whole and made architectural changes to the package to ensure its compatibility with current software development standards. Among other things, we (i) modernized the codebase to Python 3, (ii) added provisions for continuous integration and development including automatic binary releases on multiple architectures, (iii) established unit tests and test suites, (iv) fixed bugs and (v) added support for improved dependency infrastructure such as autodetection of SAM/BAM/CRAM file type via *HTSlib* (Bonfield *et al.*, 2021). All aforementioned changes were carried out without compromising the efficiency of *HTSeq*, which stems from a cross-language design via Cython (Behnel *et al.*, 2011) and SWIG (Beazley, 2003).

In conclusion, *HTSeq* 2.0 is a fast and reliable Python library for not only analysing high-throughput sequencing data, but also for quantifying gene expression from bulk and single-cell RNA-Seq experiments. Compared with the previous implementation, we added specific support for single-cell experiments and a richer API including a new data structure for managing 'islands-of-data' sparsity, improved API documentation and tutorials, fixed a number of bugs, established a robust testing and deployment framework to ensure scientific reproducibility, and enable continuous code integration. We believe these improvements will make *HTSeq* 2.0 a convenient tool for exploring and quantifying high-throughput sequencing experiment results across multiple omic modalities.

Acknowledgements

The authors thank Wofgang Huber for scientific exchanges and all HTSeq contributors for their valuable time.

Funding

This work was partially supported through a long-term European Molecular Biology Organization Fellowship [ALTF 269–2016 to F.Z.] and grant GNT1200271 from the National Health and Medical Research Council [to J.E.P.].

Conflict of Interest: none declared.

References

Anders,S. *et al.* (2015) HTSeq—a Python framework to work with high-throughput sequencing data. *Bioinformatics*, **31**, 166–169.

Beazley,D.M. (2003) Automated scientific software scripting with SWIG. *Fut. Generat. Comput. Syst. FGCS*, **19**, 599–609.

Behnel,S. *et al.* (2011) Cython: the best of both worlds. *Comput. Sci. Eng.*, **13**, 31–39.

Bonfield,J.K. *et al.* (2021) HTSlib: C library for reading/writing high-throughput sequencing data. *GigaScience*, **10**, giab007.

Harris,C.R. *et al.* (2020) Array programming with NumPy. *Nature*, **585**, 357–362.

Picelli,S. *et al.* (2013) Smart-seq2 for sensitive full-length transcriptome profiling in single cells. *Nat. Methods*, **10**, 1096–1098.

Ryan,D. *et al.*; asellappenIBM. (2021) deeptools/pyBigWig: 0.3.18. https://doi.org/10.5281/zenodo.4515486.

Virtanen,P. *et al.*; SciPy 1.0 Contributors. (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods*, **17**, 261–272.

Wolf,F.A. *et al.* (2018) SCANPY: large-scale single-cell gene expression data analysis. *Genome Biol.*, **19**, 15.

Zanini,F. *et al.* (2018) Single-cell transcriptional dynamics of flavivirus infection. *eLife*, **7**, e32942.