PLOS ONE

RESEARCH ARTICLE

Portable BLAST-like algorithm library and its implementations for command line, Python, and R

Steven Schmid[1], **Aditya Jeevannavar**[2], **Timothy R. Julian**[3], **Manu Tamminen**[2]*

1 hakuna AG, Zürich, Switzerland, **2** Department of Biology, University of Turku, Turku, Finland, **3** Department of Environmental Microbiology, Swiss Federal Institute of Aquatic Science and Technology (Eawag), Dübendorf, Switzerland

* mavatam@utu.fi

Citation: Schmid S, Jeevannavar A, Julian TR, Tamminen M (2023) Portable BLAST-like algorithm library and its implementations for command line, Python, and R. PLoS ONE 18(11): e0289693. https://doi.org/10.1371/journal.pone.0289693

Editor: M. Sohel Rahman, Bangladesh University of Engineering and Technology, BANGLADESH

Received: September 21, 2022

Accepted: July 25, 2023

Published: November 30, 2023

Data Availability Statement: The source code for nsearch, npysearch and Blaster is released under the BSD-3 license and available on Github at https://github.com/stevschmid/nsearch, https://github.com/tamminenlab/npysearch and https://github.com/tamminenlab/blaster, respectively. Furthermore, nsearch and npysearch are available on Conda-forge at https://anaconda.org/conda-forge/nsearch and https://anaconda.org/conda-forge/npysearch, respectively. Blaster is available for installation through CRAN at https://CRAN.R-project.org/package=blaster. npysearch is available

Abstract

Basic local-alignment search tool (BLAST) is a versatile and commonly used sequence analysis tool in bioinformatics. BLAST permits fast and flexible sequence similarity searches across nucleotide and amino acid sequences, leading to diverse applications such as protein domain identification, orthology searches, and phylogenetic annotation. Most BLAST implementations are command line tools which produce output as comma-separated values files. However, a portable, modular and embeddable implementation of a BLAST-like algorithm, is still missing from our toolbox. Here we present nsearch, a command line tool and C++11 library which provides BLAST-like functionality that can easily be embedded in any application. As an example of this portability we present Blaster which leverages nsearch to provide native BLAST-like functionality for the R programming language, as well as npysearch which provides similar functionality for Python. These packages permit embedding BLAST-like functionality into larger frameworks such as Shiny or Django applications. Benchmarks show that nsearch, npysearch, and Blaster are comparable in speed and accuracy to other commonly used modern BLAST implementations such as VSEARCH and BLAST+. We envision similar implementations of nsearch for other languages commonly used in data science such as Julia to facilitate sequence similarity comparisons. Nsearch, Blaster and npysearch are free to use under the BSD 3.0 license and available on Github Conda, CRAN (Blaster) and PyPi (npysearch).

Introduction

Basic local-alignment search tool (BLAST) is a versatile sequence analysis algorithm which permits fast sequence similarity searches across DNA, RNA, and amino acid sequences. BLAST searches are used in diverse applications such as sequence similarity searching [1], protein domain identification [2], orthology searches, [3] and phylogenetic annotation [4].

Common implementations of BLAST include the popular web interface provided by NCBI [1] as well as standalone command-line tools such as BLAST+ [5], usearch, [6] and VSEARCH

for installation from PyPi at https://pypi.org/project/npysearch/. The datasets generated and analysed during the current study are available at Zenodo, https://zenodo.org/record/4804623.

Funding: This work was funded by the Bill and Melinda Gates Foundation (https://www.gatesfoundation.org/) through grant OPP1151041 granted to Timothy Julian and Manu Tamminen. The foundation had no role in the study design, data collection, interpretation of the results, or submission of the work for publication.

Competing interests: The authors have declared that no competing interests exist.

[7]. In a typical use case of BLAST through the NCBI web interface, the BLAST input is submitted through a web form and the results are visually inspected using a web browser. In high-performance applications, BLAST is typically used as a Unix command line application and the output is saved into a comma-separated values table, to be processed further using programming languages such as R or Python. To facilitate this transport of data between the command line application and these programming languages, several wrappers are available such as rBLAST [8] or metablastr [9]. These wrappers require separate installation of the command line application which limits the portability and modular use of BLAST in larger processing pipelines.

To provide a portable BLAST-like algorithm, we present here nsearch which is a dependency-free C++11 library and command line application. C++ is suitable for high-performance applications and is supported as an extension language by several other programming languages. As concrete examples of the portability of nsearch, we introduce Blaster which implements a BLAST-like algorithm in R programming language [10] by embedding nsearch through Rcpp [11], and npysearch which provides similar implementation of nsearch on Python programming language [12] through pybind11 [13]. We envision similar implementations of nsearch for other languages commonly used in data science such as Julia. nsearch is light-weight, has no external dependencies, and currently supports similarity searches of DNA, RNA, and amino acid sequences.

Materials and methods

Alphabets

To distinguish between different types of sequences, nsearch uses the concept of alphabets. Alphabets define how biological sequences and their letters are treated. nsearch currently supports two alphabets: nucleic acids (DNA and RNA) and amino acids (defaulting to BLOSUM62 scoring matrix [14]), which are implemented via C++ templates. This enables straightforward implementation of custom alphabets or different scoring schemes for future releases.

nsearch database indexing and searching

nsearch incorporates database searching inspired by usearch [6] which is based on the idea that a few good candidate sequences are sufficient for most sequence similarity searches. Candidate sequences are selected from a database based on the number of unique words or k-mers (the collection of which is designated U) they share and are checked in decreasing order of U for their sequence similarity. If the similarity exceeds a user-defined threshold, the candidate is considered a hit. If the similarity is below the threshold, the candidate is considered a reject and the next best candidate is checked, until the maximum number of allowed rejects for this query has been reached.

Database searching in nsearch starts with indexing during which the database sequences are stored in lookup tables. The search algorithm will use the lookup tables to quickly identify shared k-mers across all database sequences. By default, nsearch uses $k = 8$ for the DNA alphabet and $k = 5$ for the Protein alphabet. Higher k leads in general to faster search run time at the price of reduced sensitivity [15]. Within nsearch, each k-mer is transformed into a compact binary format (32-bit unsigned integer), described by the BitMapPolicy of the alphabet (e.g. 2 bits are used to encode all DNA nucleotides).

K-mers which contain at least one ambiguous residue are mapped into a special k-mer called AmbiguousKmer. Storing each possible realization of an ambiguous k-mer is infeasible due to exponential growth and corresponding burden on memory consumption and decrease

in lookup efficiency. For this reason, regions containing ambiguous nucleotides are discarded in nsearch during database indexing.

Based on the principle that similar sequences tend to have more k-mers in common [16], the search algorithm uses the indexed database to search for candidate sequences by counting the number of unique k-mers (*U*) they share with the query sequence. The sorted list of high-scoring candidates is then checked in decreasing order of *U*: the database sequence which shares the highest number of unique k-mers with the query sequence is checked first. For this purpose, the pairwise sequence alignment is determined. The sequence alignment is used to calculate the sequence identity, which in turn is used to determine whether the candidate is a hit.

The sequence alignment algorithm determines HSPs (high-scoring segment pairs) by employing a seed-and-extend strategy [17]. The HSPs are subsequently joined into a chain in a greedy manner: The highest scoring HSP is combined with the next-best non- overlapping HSP, until there are no HSPs left to join. The missing alignments between the HSPs are determined with banded dynamic programming [18].

nsearch R and Python interfaces: Blaster and npysearch

Blaster exposes the nsearch BLAST-like functionality to R through Rcpp [11] and to Python through pybind11 [13]. These implementations expose two functions: *read_fasta* which is a utility function for parsing Fasta files into data frames or dictionaries (for R and Python, respectively), and *blast* which is an interface to the underlying nsearch BLAST implementation. *Blast* accepts the following arguments: maximum accepted results, maximum rejected results, minimum hit identity, alphabet, the searched strands, and an option to save the results into a temporary file.

Benchmarking

The benchmark data set was created using the Rfam 13.0 database [19]. The Rfam database is an annotated collection of non-coding RNA (ncRNA) families and is based on multiple sequence alignments. The ncRNAs are divided into families based on evolution from a common ancestor and the sequences are annotated with taxonomic identity of the host organism. The division into families, the taxonomic annotation, as well as the high sequence diversity permits using Rfam to benchmark the accuracy of the searches of nsearch and other tools.

The following search settings were used: (1) report one hit (which has at least a 75% sequence identity between query and database sequence), (2) terminate after 8 rejects (high scoring candidates which have below 75% sequence identity), (3) use all available CPU cores, (4) no masking.

The following hardware was used: Macbook Pro M1, 2020, 16 GB Memory, running macOS Monterey 12.5.

Results

For every Rfam family which contains 2 or more sequences, one sequence was selected randomly for the query set (resulting in 2,085 sequences), with the remainder in the reference database (380,796 sequences). Given the high sequence diversity of the Rfam database, the search algorithm should ideally recoved a sequence from the database corresponding to the query sequence. The accuracy of the search was therefore calculated as the proportion of the matches where the database match corresponded to the query sequence. nsearch performance was directly comparable with VSEARCH and BLAST+, both in run time and accuracy

Table 1. Search performance comparison of nsearch to other tools. Mean run time is calculated from 10 replicates, using a query set of 2,085 sequences against a database of 380,796 sequences from Rfam. Accuracy is calculated as the percentage of matches between the query and database target sequences.

Software	Mean run time (s)	Accuracy	Notes
BLAST+	1.4	98.1%	The legacy BLAST algorithm
VSEARCH	0.88	98.7%	Another, non-modular implementation of the BLAST algorithm [7]
nsearch	1.52	98.3%	Modular implementation of the BLAST algorithm (this study)
Blaster	2.63	98.3%	Implementation of nsearch on R using Rcpp (this study)
npysearch	1.43	98.3%	Implementation of nsearch on Python using pybind11 (this study)

https://doi.org/10.1371/journal.pone.0289693.t001

(Table 1). The code and data for running the benchmarks is available at https://zenodo.org/record/4804623.

Discussion

nsearch is an efficient, dependency-free implementation of a BLAST-like algorithm, written in C++11. Benchmarking nsearch against the commonly used general purpose tools, VSEARCH [7] and BLAST+ [5], demonstrates that nsearch can compete in run time and accuracy with comparable state-of-the-art tools. To demonstrate the portability of nsearch, we present here Blaster and npysearch, implementations of a BLAST-like algorithm on R and Python programming languages, respectively, powered by nsearch. This portability will permit implementations of nsearch into other languages commonly used in data science such as Julia or Ruby, and into larger software frameworks such as Django [20] or Shiny [21] applications. The source code for nsearch, npysearch and Blaster is released under the BSD-3 license and available on Github at https://github.com/stevschmid/nsearch, https://github.com/tamminenlab/npysearch and https://github.com/tamminenlab/blaster, respectively. Furthermore, nsearch, Blaster, and npysearch are available on Conda-forge at https://anaconda.org/conda-forge/nsearch, https://anaconda.org/conda-forge/r-blaster and https://anaconda.org/conda-forge/npysearch, respectively. Blaster is also available on CRAN at https://CRAN.R-project.org/package=blaster and npysearch on PyPi at https://pypi.org/project/npysearch/. The datasets generated and analysed during the current study are available at Zenodo, https://zenodo.org/record/4804623.

Author Contributions

Conceptualization: Timothy R. Julian, Manu Tamminen.

Funding acquisition: Timothy R. Julian.

Software: Steven Schmid, Aditya Jeevannavar, Manu Tamminen.

Supervision: Timothy R. Julian, Manu Tamminen.

Validation: Steven Schmid.

Writing – original draft: Steven Schmid, Timothy R. Julian, Manu Tamminen.

Writing – review & editing: Manu Tamminen.

References

1. Johnson M, Zaretskaya I, Raytselis Y, Merezhuk Y, McGinnis S, Madden TL. NCBI BLAST: a better web interface. Nucleic Acids Res. 2008; 36(Web Server issue).

2. George RA, Heringa J. Protein domain identification and improved sequence similarity searching using PSI-BLAST. Proteins: Structure, Function, and Genetics. 2002 Sep 1; 48(4):672–81. https://doi.org/10.1002/prot.10175 PMID: 12211035
3. Zhou Y, Landweber LF. BLASTO: A tool for searching orthologous groups. Nucleic Acids Res. 2007 Jul 1; 35(SUPPL.2):W678–82. https://doi.org/10.1093/nar/gkm278 PMID: 17483516
4. Hanekamp K, Bohnebeck U, Beszteri B, Valentin K. PhyloGena a user-friendly system for automated phylogenetic annotation of unknown sequences. Bioinformatics. 2007 Apr 1; 23(7):793–801. https://doi.org/10.1093/bioinformatics/btm016 PMID: 17332025
5. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, et al. BLAST+: Architecture and applications. BMC Bioinformatics. 2009 Dec 15; 10(1):421. https://doi.org/10.1186/1471-2105-10-421 PMID: 20003500
6. Edgar RC. Search and clustering orders of magnitude faster than BLAST. Bioinformatics. 2010 Oct 1; 26(19):2460–1. https://doi.org/10.1093/bioinformatics/btq461 PMID: 20709691
7. Rognes T, Flouri T, Nichols B, Quince C, Mahé F. VSEARCH: A versatile open source tool for metagenomics. PeerJ [Internet]. 2016 Oct 18 [cited 2021 Mar 18]; 2016(10):e2584. Available from: https://github.com/torognes/vsearch
8. mhahsler/rBLAST: Interface for the Basic Local Alignment Search Tool (BLAST)—R-Package [Internet]. [cited 2021 Aug 5]. Available from: https://github.com/mhahsler/rBLAST
9. Benoit M, Drost HG. A Predictive Approach to Infer the Activity and Natural Variation of Retrotransposon Families in Plants. Methods in Molecular Biology. 2021; 2250:1–14. https://doi.org/10.1007/978-1-0716-1134-0_1 PMID: 33900588
10. R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria; 2020.
11. Eddelbuettel D, François R. Rcpp: Seamless R and C++ integration. J Stat Softw. 2011 Apr 13; 40 (8):1–18.
12. van Rossum G, Drake FL Jr. Python reference manual. Centrum voor Wiskunde en Informatica Amsterdam; 1995.
13. Jakob W, Rhinelander J, Moldovan D. pybind11—Seamless operability between C++11 and Python. 2016.
14. Henikoff S, Henikoff JG. Amino acid substitution matrices from protein blocks. Proc Natl Acad Sci U S A. 1992; 89(22):10915–9. https://doi.org/10.1073/pnas.89.22.10915 PMID: 1438297
15. McGinnis S, Madden TL. BLAST: at the core of a powerful and diverse set of sequence analysis tools. Nucleic Acids Res [Internet]. 2004 Jul 1 [cited 2023 Jul 18]; 32(suppl_2):W20–5. Available from: https://doi.org/10.1093/nar/gkh435 PMID: 15215342
16. Edgar RC. Local homology recognition and distance measures in linear time using compressed amino acid alphabets. Nucleic Acids Res. 2004 Jan 1; 32(1):380–5. https://doi.org/10.1093/nar/gkh180 PMID: 14729922
17. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990 Oct 5; 215(3):403–10. https://doi.org/10.1016/S0022-2836(05)80360-2 PMID: 2231712
18. Chao KM, Pearson WR, Miller W. Aligning two sequences within a specified diagonal band. Bioinformatics. 1992 Oct 1; 8(5):481–7. https://doi.org/10.1093/bioinformatics/8.5.481 PMID: 1422882
19. Kalvari I, Nawrocki EP, Argasinska J, Quinones-Olvera N, Finn RD, Bateman A, et al. Non-Coding RNA Analysis Using the Rfam Database. Curr Protoc Bioinformatics. 2018 Jun 1; 62(1):e51. https://doi.org/10.1002/cpbi.51 PMID: 29927072
20. Django Software Foundation. Django [Internet]. 2022 [cited 2022 Dec 12]. Available from: https://djangoproject.com
21. Chang W, Cheng J, Allaire JJ, Sievert C, Schloerke B, Xie Y, et al. shiny: Web Application Framework for R [Internet]. 2022. Available from: https://CRAN.R-project.org/package=shiny