iBLAST: intelligent BLAST of new sequences via automated e-value correction

Sajal Dash[1*], Sarthok Rasique Rahman[2], Heather M. Hines[2,3], Wu-chun Feng[1,4,5,6*]

1 Department of Computer Science, Virginia Tech, Blacksburg, VA, USA

2 Department of Biology, The Pennsylvania State University, University Park, PA, USA

3 Department of Entomology, The Pennsylvania State University, University Park, PA, USA

4 Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA

5 Department of Biomedical Engineering and Mechanics, Virginia Tech, Blacksburg, VA, USA

6 Health Sciences, Virginia Tech, Blacksburg, VA, USA

* sajal@vt.edu, feng@cs.vt.edu

Abstract

Search results from local alignment search tools use statistical scores that are sensitive to the size of the database to report the quality of the result. For example, NCBI BLAST reports the best matches using similarity scores and expect values (i.e., e-values) calculated against the database size. Given the astronomical growth in genomics data throughout a genomic research investigation, sequence databases grow as new sequences are continuously being added to these databases. As a consequence, the results (e.g., best hits) and associated statistics (e.g., e-values) for a specific set of queries may change over the course of a genomic investigation. Thus, to update the results of a previously conducted BLAST search to find the best matches on an *updated* database, scientists must currently rerun the BLAST search against the *entire* updated database, which translates into irrecoverable and, in turn, wasted execution time, money, and computational resources. To address this issue, we devise a novel and efficient method to redeem past BLAST searches by introducing iBLAST. iBLAST

intelligently leverages previous BLAST search results to conduct the same query search but only on the incremental (i.e., newly added) part of the database, recomputes the associated critical statistics such as e-values, and combines these results to produce updated search results. Our experimental results and fidelity analyses show that iBLAST delivers search results that are identical to NCBI BLAST at a substantially reduced computational cost, i.e., iBLAST performs $(1 + \delta)/\delta$ times faster than NCBI BLAST, where δ represents the fraction of database growth. We then present three different use cases to demonstrate that iBLAST can enable efficient biological discovery at a much faster speed with a substantially reduced computational cost.

Availability: iBLAST is available at the following web site:

`https://github.com/vtsynergy/iBLAST`

Keywords: Pairwise sequence search, sequence similarity search, BLAST, e-value correction, taxon-specific BLAST, local alignment

Contacts: `sajal@vt.edu`, `wfeng@vt.edu`

Supplementary information: Supplementary data is available at the end of the document.

Author Summary

Local alignment search is a critical step towards establishing sequence identity in many bioinformatic-analyses. The databases of biological sequences that these analyses query grow exponentially over time and a bioinformatic analysis can run for a lengthy period of time. Hence, an incremental update of the search-result can expedite the analysis by lowering the required compute resources. For a large query such as a new organism's whole transcriptome, this requirement can be financially and chronologically prohibitive. Traditional sequence similarity search tools such as NCBI BLAST performs searches from scratch every time a database accumulates new sequences. Incrementally updating search results requires merging two search results from two subsets of the same database using statistical correction. In this work, we developed necessary statistical methods for correcting e-value (a quality measure of the search-result) when two or more search results are combined and developed a sequence similarity search tool called iBLAST that leverages these e-value correction methods to perform incremental searches. Through

three different case studies representing typical scenarios encountered in bioinformatics research, we demonstrate that iBLAST finds better search-results than NCBI BLAST on a novel and large transcriptomic dataset at a much-reduced computational cost.

Introduction

The utilization of a sequence similarity search tool is an indispensable step in most bioinformatics research involving nucleotide or protein sequences. *BLAST*, short for Basic Local Alignment Search Tool, is a widely used tool capable of conducting a sequence similarity search for a sequence of interest against a sequence database. BLAST relies on a heuristic approach for searching and provides results based on the identification of regions of similarity between target and query sequences through a seed-and-extend based local alignment [1]. The number of queries and the size of the reference database can significantly impact the execution time of BLAST. Hence, the fast accumulation of sequences in NCBI-curated databases have a profound impact on the computational efforts required to perform sequence similarity searches. The sequencing data that is stored in the NCBI database has grown tremendously over the years, reportedly doubling the number of bases submitted to GenBank [2] every year over the last three decades (1982-present). This rapid accumulation of sequence data is one of the key factors responsible for transforming the field of genomics into one of the most demanding big-data science disciplines [3].

Hence, providing fast and biologically valuable sequence alignment tools via high-performance computing (HPC) and algorithmic innovations has been a highly active area of bioinformatics research, particularly in the context of rapidly expanding databases. For example, several sequence alignment programs have relied on contributing algorithmic improvements (e.g., HMMER [4], DIAMOND [5], CaBLAST [6]) while others have focused on improving parallelization to take advantage of emerging high-performance computing (HPC) platforms and programming paradigms (e.g., cuBLASTP [7], muBLASTP [8], mpiBLAST [9], SparkBLAST [10], and SparkLeBLAST [11]).

Both DIAMOND [5] and CaBLAST [6] improve the execution time of sequence alignment by compressing the sequence database. Specifically, DIAMOND reduces the

amino-acid alphabet while CaBLAST compresses the sequences by sequence redundancy. All of these sequence similarity search tools improve computational speed (i.e., reduce execution time) but sometimes at the cost of reduced sensitivity. DIAMOND only achieves 91%-99% sensitivity while CaBLAST achieves more than 99% sensitivity.

Sequence similarity tools play a vital role in genome projects as annotations of assembled sequences require the utilization of BLAST-like tools for homology assignment. In reality, genome sequencing and annotation projects can be fairly long term, and thus, can require multiple sequence updates, e.g., regular annotation updates [12, 13]. However, such updates require executing sequence similarity search from scratch as BLAST uses similarity scores and e-values that depend on the ever-increasing size of the database. For this reason, it is currently required to discard the results of prior search efforts and rerun the entire search, which translates to unredeemable execution time, money, and computational resources, as shown in Fig. 1(A). For bioinformatics projects requiring large-scale sequence similarity searches, such as those involving many transcriptomes from many taxa, the computational burden can be especially prohibitive. This problem could be addressed by performing iterative taxon-specific searches rather than conducting BLAST on the entire non-redundant (nr) database. However, adopting such an approach has been historically difficult as one would need to standardize e-values while adding new databases to find the optimal identity of each query, as shown in Fig. 1(B).

In practice, new sequences get added to the search database(s) of interest in two ways: temporally or spatially. Adding new sequences *temporally*, as illustrated in Fig. 1(A), means that new sequences are added to a database over time (e.g., a regular update to the nr database). Adding new sequences *spatially*, as illustrated in Fig. 1(B), means that different databases are available for search simultaneously and that we need to combine the search results against these databases as if the result was obtained by searching against a combination of all these databases. Currently, to the best of our knowledge, there exists no tool that can merge BLAST results of databases that have been added to either temporally or spatially. Thus, we propose *a statistical approach to compose temporal and spatial BLAST search results through a novel method of e-value correction*. We derive how to do this mathematically and provide a software artifact called iBLAST to implement this automated e-value correction.

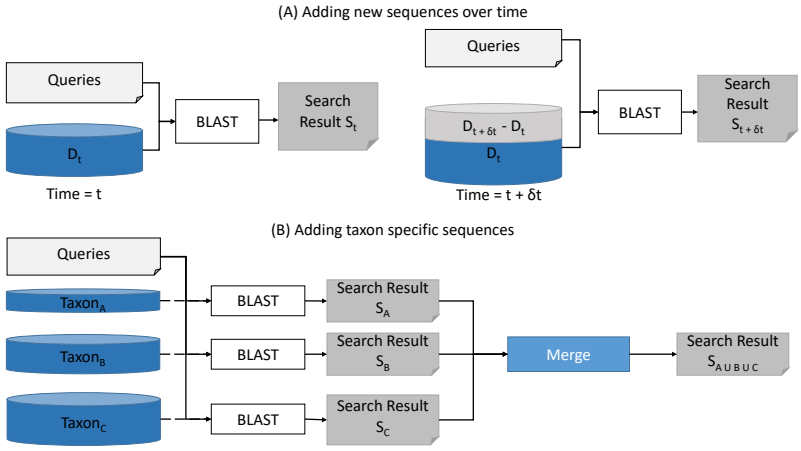


Fig 1. Addition of new sequences. (A) BLAST search when new sequences are added to the database. At time t, the database is D_t. In next δt interval, new sequences $D_{t+\delta t} - D_t$ are added, and the database becomes $D_{t+\delta t}$. With the traditional approach, the prior search result at time t cannot be reused, and we have to perform an entire BLAST search against the entire $D_{t+\delta t}$ database. (B) BLAST search when several taxon-specific databases are present and a result against the combined database is needed. For three taxa, A, B, and C, we can perform individual BLAST searches against the databases D_A, D_B, D_C, respectively. If we want to obtain a search result against the combined database $D_{A\cup B\cup C}$, we need to merge the search results in a way that their e-values reflect the combined database size.

In summary, iBLAST facilitates the recycling of previous BLAST search results and subsequently saves a substantial amount of execution time and computational resources. It also enables taxon-specific BLAST searches, including the incremental addition of specific biologically-relevant taxa to BLAST databases with subsequent merging. The iBLAST tool consists of Python modules that are compatible with all recent versions (since 2012) of the NCBI BLAST command-line tools. It can run on all major operating systems (Windows, Linux, and MacOS). The iBLAST tool is particularly useful for bioinformatics analysis projects involving large-scale sequence similarity search tasks. Furthermore, it incurs minimal cognitive and installation overhead for the users of NCBI BLAST.

Our contribution

We report three main contributions in this paper.

1. We devised e-value correction formulas for Karlin-Altschul and Spouge statistics to combine BLAST search results performed on different sequence databases across various time instances and domains.

2. We created *iBLAST*, a sequence similarity search tool that allows search against a growing sequence database through automated e-value correction at $(1 + \delta)/\delta$ faster speed than NCBI BLAST, where δ represents the fraction of new sequences that have been added to the sequence database.

3. We demonstrated the efficacy and means to incorporate domain knowledge into bioinformatics analysis using iBLAST through case studies using *de novo* assembled transcriptome of the venom gland of an Oak gall wasp.

Results

We created a tool called iBLAST for e-value correction and incremental BLAST search in the temporal and spatial domain. Our first result is the tool itself, which we describe below, along with its usage. Then, through case study I, we show that bioinformaticians can use iBLAST for e-value correction and incremental BLAST search for moderate query sizes via a typical BLAST procedure. Through case study II, we show that iBLAST can achieve $(1 + \delta)/\delta$ speedup over NCBI BLAST for a δ increase in the database; for a large query, this speedup can be significant. Via case study III, we demonstrate that incorporating taxon-specific domain knowledge and our methods of e-value correction can reduce BLAST search space significantly.

iBLAST software allows the user to perform incremental BLAST search with minimal overhead

The iBLAST program v1.0 includes a collection of Python scripts that can be downloaded at `https://github.com/vtsynergy/iBLAST`. To facilitate the use of iBLAST, we have provided instructions for installation, a user's manual, a quick start guide and examples of how to use iBLAST in several typical scenarios of bioinformatics research.

First, the user needs to copy the source folder and run the following command from this directory to install iBLAST:

```
./iBLAST-installer.sh
```

Next, we move onto the Python scripts, as described below.

Main program: iBLAST.py This program provides intelligent and incremental BLAST search options. It takes in a regular BLAST search command and performs an incremental search. Below is an example of how to use the script.

```
python iBLAST.py "blastp -db nr -query Trinity-tx.fasta -outfmt 5 -out result.xml"
```

Merge scripts: These scripts merge the results of two BLAST searches in XML format and produce an XML output with corrected e-values.

1. **BlastpMergerModule.py:** This script merges the results obtained using Karlin-Altschul statistics (e.g., blastn results).
2. **BlastnMergerModule.py:** This script merges the results obtained using Spouge statistics (e.g., blastp results).
3. **BlastpMergerModuleX.py** These scripts merge more than two BLAST results. They require several results to merge the input and output.

```
python BlastpMergerModule.py input1.xml input2.xml output.xml
python BlastnMergerModule.py input1.xml input2.xml output.xml
python BlastpMergerModuleX.py 3 input1.xml input2.xml input3.xml output.xml
```

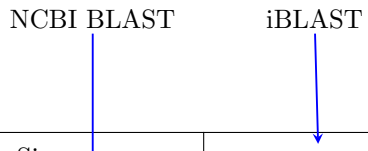
Case study I: method verification and performance

Verification. In case study I, we validate whether we can achieve the same results from a single NCBI BLAST search as from the iBLAST. As shown in Table 1 and Table 2, iBLAST delivers the same results as NCBI BLAST with a 100% e-value match and 100% hit match.

- **blastn**: Sequence alignment using blastn was performed on nt databases (nucleotide sequences). In all three time periods, iBLAST finds all the same hits and in the same order as NCBI BLAST does for blastn, including $3,964$ hits at time t_0; $4,150$ hits at time t_1, and $4,924$ hits at time t_2, thus validating iBLAST with respect to Karlin-Altschul statistics (Table 1).
- **blastp**: Sequence alignment using blastp was performed on nr (non-redundant protein sequences) databases. For each of these three time periods, iBLAST

reports the same hits in the same order as NCBI BLAST for blastp. The numbers of reported hits in these three time periods for blastp are 45, 154 hits; 46, 356 hits; and 46, 869 hits, respectively, thus validating iBLAST with respect to Spouge statistics (Table 2).

Table 1. Case study I: Fidelity of iBLAST in three consecutive time periods. *blastn* search was performed on nucleotide sequence databases (nt). At any time instance, the *Past* database size is the size of the database from the previous time instance. The *Present* database size is the database size at the present time instance. *Delta* is the incremental database growth from the previous time instance to the current time instance. NCBI BLAST must be performed on the entire *Present* database size, while iBLAST only needs to be performed on *Delta*.



Time	Search	Data-base	Database Size		Delta = Present − Past	e-value Match	Hit Match
			Past	Present			
t_0	blastn	nt	0	80,740,533,243	80,740,533,243	100%	100%
t_1	blastn	nt	80,740,533,243	113,749,495,340	33,008,962,097	100%	100%
t_2	blastn	nt	113,749,495,340	152,471,828,601	38,722,333,261	100%	100%

Performance. For a δ increase in database size, iBLAST performs $(1 + \delta)/(\delta)$ times faster than NCBI BLAST. Fig. 2 shows the time saved for both blastp and blastn, respectively, using iBLAST, resulting in a speedup ranging between approximately three- and five-fold.

Case study II: large-scale alignment tasks on novel datasets

We performed searches using iBLAST and NCBI BLAST, where a newly obtained gall wasp (Hymenoptera: Cynipidae) transcriptome dataset was utilized as queries in two time periods across which there was a 48% increase in the nr database (Table 3). In both time periods, iBLAST reports the same hits in the same order as an NCBI BLAST run.

Table 2. Case study I: Fidelity of iBLAST in three consecutive time periods. *blastp* search was performed on nucleotide protein databases (nr). At any time instance, the *Past* database size is the size of the database from the previous time instance. The *Present* database size is the final database size at that instance. *Delta* is the incremental database growth from previous time instance to the current time instance. NCBI BLAST is performed on the *Present* database size, while iBLAST is performed only on *Delta*.

NCBI BLAST iBLAST

Time	Search	Data-base	Database Size		Delta = Present – Past	e-value Match	Hit Match
			Past	Present			
t_0	blastp	nr	0	17,686,779,866	17,686,779,866	100%	100%
t_1	blastp	nr	17,686,779,866	23,752,080,639	6,065,300,773	100%	100%
t_2	blastp	nr	23,752,080,639	30,030,148,449	6,278,067,810	100%	100%

Table 3. Case study II: Fidelity of iBLAST in two consecutive time periods. *blastp* search was performed on a large-scale novel transcriptome. At any time instance, the *Past* database size is the size of the database from the previous time instance. The *Present* database size is the final database size at that instance. *Delta* is the incremental database growth from previous time instance to the current time instance. NCBI BLAST is performed on the Present database size, while iBLAST is performed only on *Delta*.

NCBI BLAST iBLAST

Time	Search	Data-base	Database Size		Delta = Present – Past	e-value Match	Hit Match
			Past	Present			
t_0	blastp	nr	0	40,077,622,077	40,077,622,077	100%	100%
t_1	blastp	nr	40,077,622,077	59,270,473,315	19,192,851,238	100%	100%

For this increase in the database size, iBLAST is 3.1 times faster than NCBI BLAST. Relative to the total execution time of 134 minutes, the time needed for e-value correction and merging the results is minimal, i.e., less than a minute using only 20 cores. Overall, NCBI BLAST completes the alignment search in 24,862 seconds (6 hours, 54 minutes) on average, while iBLAST completes the search in only 8,009 seconds (2

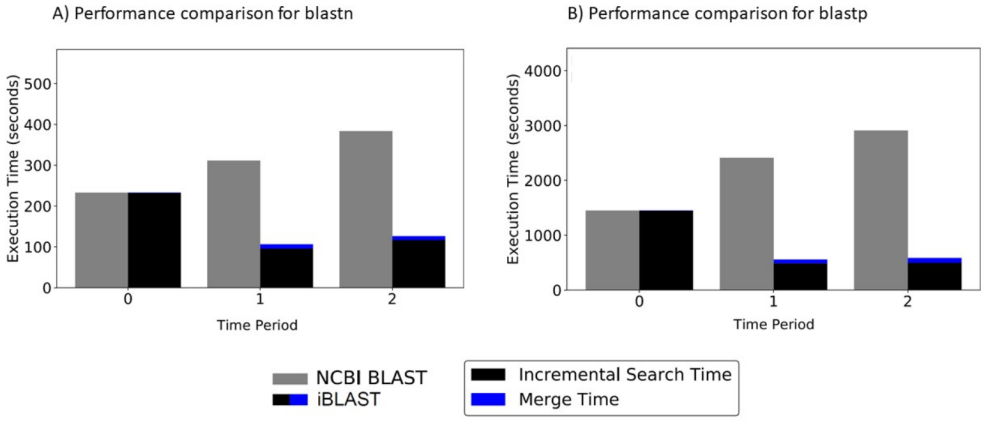


Fig 2. Performance comparison between NCBI BLAST and iBLAST for case study I. (A) Performance comparison between regular blastn and incremental blastn at 3 periods when nt database is growing over time, using 100 nucleotide queries. For 40.8% and 34.0% increase in the database size, iBLAST performs 2.93 and 3.03 times faster respectively. (B) Performance comparison between regular blastp and incremental blastp at 3 periods when nr database is growing over time, using 100 protein queries. For 34.1% and 26.3% increase in the database size, iBLAST performs 4.33 and 4.98 times faster respectively.

hours, 14 minutes). The merge time for each of these tasks is 40 seconds on average. This computational efficiency matches our projected speedup $(1 + 0.48)/0.48 = 3.08$.

We observed the effect of query partitioning on load balancing (see Section "Distributing workload across nodes"). Our approach to partition the queries based on the number of residues shows superior load-balancing over the traditional strategy to partitioning the queries based on the number of queries. We elaborate on this point further in Section "Load-balancing via query partitioning."

Case study III: Taxon-specific searches to expedite informatics

To examine the fidelity of iBLAST while merging multiple (taxon-specific) databases, we first compared the iBLAST merged results from multiple individual BLAST (blastp) searches on seven biologically relevant taxa separately to results obtained when a BLAST search was performed against a database combining all the sequences belonging to these taxa simultaneously. The result exhibits 100% fidelity. Then, as presented in Table 4, we compare the merged BLAST results of individual taxon-level database search with the BLAST results obtained in case study II (time period 1), where the same queries were searched against the entire nr database to better understand the

relative time savings vs. accuracy of taxon-guided approaches. The taxon-specific approach is much more time-efficient and computationally inexpensive as it searched much smaller-sized databases. With our initial set of 6 taxa, we sampled only 0.35% of the nr database and retrieved 8.124% of the top hits obtained when searching nr. Although this number is low, the identity of top hits is likely to be similar even if the best taxonomic hit to that gene was not retrieved, as gall wasps do not have any close relatives currently hosted in NCBI databases but rather many equidistant relatives. Given this, we then added in sequences of the rest of Hymenopteran species to see if this improves the number of shared top hits. With this analysis, we conducted BLAST search on only 1.17% of the total nr yet obtained 87.75% similarity in top hits to a full nr BLAST. This result demonstrates the potential of performing more taxon-guided approaches to save on the costs of large-scale BLAST searching jobs. Performing the analysis in this way has also enabled improved curation of hits by taxon, which facilitates better biological interpretation of these results.

iBLAST finds better scoring hits that are missed by NCBI BLAST

While iBLAST finds all the hits reported by NCBI BLAST in the same order of appearance, iBLAST reports several *better scoring hits* that NCBI BLAST misses in all the case studies. Since case study II covers the most number of hits, we quantified these missed hits for this case study. NCBI BLAST misses 1.57% (13171 out of 837942 top hits) of the better scoring hits. Command-line NCBI BLAST uses a search parameter *max_target_seqs* in an unintended way where instead of reporting all the *best max_target_seqs* hits, it has a bias toward *first max_target_seqs* hits. A comprehensive discussion about this issue was carried out by Sujai Kumar (`https://gist.github.com/sujaikumar/504b3b7024eaf3a04ef5/`) and two other teams of researchers [14, 15]. In this process, it misses some of the better scoring hits that are discovered in a later phase of the search. (Details can be found in Section "Explanation for NCBI BLAST missing many top hits.") This is an extra advantage of iBLAST over NCBI BLAST. Since the former works on smaller databases and then combines the results instead of searching a single large database, it has more

Table 4. Potential for taxon-guided searches enabled by iBLAST. Comparison of merged BLAST results from multiple individual BLAST searches with a separate BLAST search conducted against a completed nr database shows that biologically relevant taxa can be added incrementally to obtain similar results to nr by searching against a much smaller database size.

Species	NCBI taxon id	%nr sequences covered	Number of nr top hits covered	%nr top hits covered
Nasonia vitripennis (jewel wasp)	7425	0.02%	853	4.84%
Apis mellifera (honey bee)	7460	0.02%	207	1.17%
Harpegnathos saltator (Jerdon's jumping ant)	10380	0.03%	347	1.96%
Drosophila melanogaster (fruit fly)	7227	0.08%	6	0.034%
Quercus suber (cork oak)	58331	0.09%	0	0.00%
Glycine max (soybean)	3847	0.11%	22	0.12%
Rest of Hymenoptera	7399	0.83%	14281	80.98%
Total	**multiple**	**1.17%**	**15476**	**87.75%**

candidate hits to choose from for reporting final hits.

Methods

To perform an iBLAST search *temporally*, we only need to consider the newly arrived sequences in the interval δt and perform a BLAST search against these sequences to get the result $S_{\delta t}$, as shown in Fig. 3(A). iBLAST then corrects the e-value scores for this incremental result $S_{\delta t}$ and the past result S_t by using the size of the database $D_{t+\delta t} = D_t + D_{\delta t}$. To perform an iBLAST search *spatially*, as shown in Fig. 3(B), iBLAST examines the search results from different databases and corrects their e-values by using the size of the combined database $D_{A \cup B \cup C} = D_A \cup D_B \cup D_C$. Then, iBLAST merges these search results with corrected e-values to obtain the final search result

$S_{A \cup B \cup C}$.

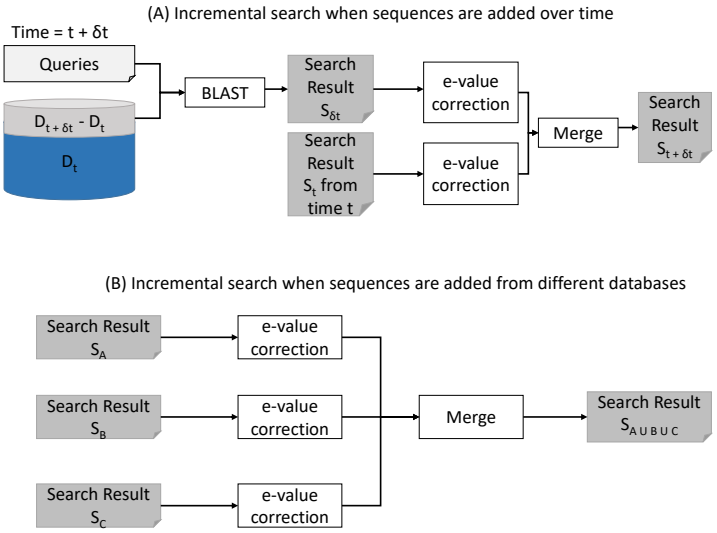


Fig 3. Incremental search by iBLAST. (A) Incremental search when new sequences get added to the database over time. We perform a BLAST search against the incremental database and combine the result with past results after e-value correction. (B) Incremental search when search results from different databases are available. Different search results are corrected for e-value against the combined database size; the corrected results are then merged together.

In the remaining part of this section, we present the details of our e-value correction methodology, the implementation details of iBLAST, and the fidelity and efficacy of iBLAST over NCBI BLAST via three case studies. We start with some background on core concepts of BLAST constructs, statistics, and metrics.

BLAST concepts and statistics

Here we present the core concepts that underlie BLAST, including the statistics for e-value computation and existing methods for correcting e-value computation when the size of the database is perceived to have changed.

Core concepts of a BLAST result: hit, HSP, score, and e-value

When we perform a BLAST search against a sequence database with a query sequence, the BLAST program returns the best matching sequences from the target database. These best-matching sequences are called *hits*. Between the query and a hit sequence, there exist many pairwise local matches, which are called *high scoring pairs* or *HSPs*.

One hit can consist of many HSPs. HSPs are scored using some statistical metrics when comparing aligned symbols. The score for a hit is the score of the highest-scoring HSP that belongs to that hit. The e-value for an HSP is computed using the score, the database size, and other statistical parameters. The reported e-value of a hit is the e-value of the highest-scoring HSP of this hit [16].

BLAST statistics for e-value computation

BLAST programs use two different types of statistics for e-value computation: Karlin-Altschul statistics and Spouge statistics. Both of these statistical formulae calculate e-value for the HSPs and hits. *blastn* and *tblastx* use Karlin-Altschul statistics while *blastp*, *blastx*, and *tblastn* use Spouge statistics.

Karlin-Altschul statistics. Karlin-Altschul statistics [1] measures the e-value using $E = e^{-\lambda(S-\mu)} = Kmne^{-\lambda S}$. This formula is adjusted for edge effect (see supplementary section). Here $m = m_a - l, n = n_a - N \times l$, and $l = \ln(K \times m \times n)/H$ while N is the number of sequences in the database, m_a is the actual length of the query, n_a is the actual length of the database, and H is the entropy of the scoring system. The length adjustment l satisfies $l = \frac{\alpha}{\lambda} \ln(K(m_a - l)(n_a - Nl)) + \beta$. Here, $\alpha, \beta, and \lambda$ are Karlin-Altschul parameters.

Spouge statistics. Spouge statistics [17] is developed on the Karlin-Altschul formula. Instead of computing the length adjustment l and then using it to compute the effective length of the database and query, Spouge statistics applies a finite-size correction (FSC). Instead of estimating l, FSC estimates $area = E[m_a - L_I(y)]^+[n_a - L_J(y)]^+$ as a measure of $(m_a - l)(n_a - Nl)$. The e-value E is then calculated as $E = area \times Ke^{-\lambda S} \times db\_scale\_factor$ where $db\_scale\_factor = \frac{n}{m}$.

Existing e-value correction software and their features

Here we discuss the different approaches for correcting e-value scores when the search database is partitioned.

mpiBLAST [9], a parallel implementation of NCBI BLAST on a cluster, partitions the database and performs BLAST searches against these partitions in parallel. For

accurate e-value correction, mpiBLAST requires prior knowledge of the entire database [9]. NOBLAST [18] corrects e-values when split databases are in use and results need to be aggregated. However, it does not work with Spouge statistics. We provide a detailed explanation of these tools in Section "Existing e-value correction software and their features."

Both tools (mpiBLAST and NOBLAST) provide exact e-value statistics for Karlin-Altschul statistics when knowledge about the entire database is available *a priori*. However, they are not useful when the database keeps changing or when two different search results against two different instances of similar databases need to be aggregated.

Table 5 provides a high-level comparison between mpiBLAST, NOBLAST, and our iBLAST.

Table 5. Comparison of three different BLAST tools that explicitly deal with e-value statistics correction. iBLAST supports e-value correction across time *and* space without requiring prior knowledge of the entire database while the other tools can perform e-value correction in limited scenarios.

Feature	mpiBLAST	NOBLAST	iBLAST
E-value correction for Karlin-Altschul statistics	✓	✓	✓
E-value correction for Spouge statistics	✗	✗	✓
Aggregate search results against pre-planned database segments	✓	✓	✓
Aggregate search results against arbitrary database instances	✗	✗	✓
Reuse existing search results	✗	✗	✓

Redundancy in data vs. redundancy in computation

Prior efforts to leverage redundancy in data (e.g., DIAMOND and CaBLAST) have successfully accelerated BLAST but at the cost of a small reduction in sensitivity. CaBLAST's compressive algorithm achieves over 99% sensitivity for the improved speed [6]. Different versions of DIAMOND have sensitivity in the range $91.04\% - 99\%$ for various datasets [5]. In contrast, iBLAST aims to eliminate redundant computation while maintaining 100% sensitivity.

e-value correction in an incremental setting

Correct e-value computation requires the actual database length (i.e., total number of bases/residues) in both Karlin-Altschul statistics and Spouge statistics. While database-partitioning parallel BLAST applications, like mpiBLAST and NOBLAST, require prior knowledge about the total database length, iBLAST leverages the partial knowledge from a previous BLAST search and combines it with the new sequence additions to the database to infer the total database length and compute the adjusted e-value in relation to the updated database. The mpiBLAST and NOBLAST tools pass the actual database length to each of their parallel jobs, thus forcing the statistics module to compute correct e-values from the beginning. For the iBLAST search, whenever new data arrives to the database, the pairwise sequence search is automatically refined in two steps. First, the search is only run on the databases constructed from *new* sequences that have been added to the database. Second, the results generated from searching the new sequences in the database are then merged with the saved results from the previous BLAST search.

e-value correction for Karlin-Altschul statistics

Let n_c represent the current database length and n_d represent the length of the newly arrived sequences for the database. Also, let N_c be the number of sequences in the current database and N_d be the number of sequences in the newly arrived part of the database. Then, we have

$$\text{Actual length of the updated database: } n_t = n_c + n_d.$$
$$\text{Total number of sequences in updated database: } N_t = N_c + N_d.$$

The actual query length m does not change with the change in the database. However, we do need to recompute the effective length l by solving the fixed-point equation for the new database length using Equation (1).

$$l = \frac{\alpha}{\lambda} \ln\Big(K(m-l)\big((n_c+n_d) - (N_c+N_d)\times l\big)\Big) + \beta \tag{1}$$

Now, with the updated length adjustment l, we can either recompute the e-values for all the matches or correct the e-values. To recompute all the e-values from scratch, we

use Equation (2).

$$E = e^{-\lambda(S-\mu)} = K(m-l)\big((n_c+n_d)-l\big)e^{-\lambda S} \tag{2}$$

Alternatively, we can correct the e-values from the current values. First, we use l to recompute the value of the effective search space. We then use the newly computed effective search space to recalibrate the e-values for all the reported HSPs from the current and delta search results. Assuming that D_{part} is the partial effective search space and that D_{total} is the total effective search space, then the corrected e-value is given by

$$E_{total} = E_{part} + Ke^{-\lambda S} \times (D_{total} - D_{part}) \tag{3}$$

While both approaches require a constant number of arithmetic operations, the former approach, i.e., recomputing all the e-values from scratch, requires fewer arithmetic operations.

e-value correction for Spouge statistics

For Spouge statistics, the value of *area* described in Section "Spouge statistics" does not change since it is a function of the query length, sequence length, and Gumbel parameters. However, the database scale factor does change, and thus, we need to account for it. If the actual database lengths for the newly added part of the database and the total database are n_{part} and n_{total}, respectively, then

$$E_{part} = area \times e^{-\lambda S} \times \frac{n_{part}}{m} \quad \text{and} \quad E_{total} = area \times e^{-\lambda S} \times \frac{n_{total}}{m}$$

So,

$$E_{total} = E_{part} \times \frac{n_{total}}{n_{part}} \tag{4}$$

Therefore, based on this derivation, we only have to re-scale the e-values instead of using Spouge's e-value computational methods.[1]

[1]Note: For re-scaling e-values that have been previously (and imprecisely) rounded to 0.0 by NCBI BLAST, re-scaling an e-value smaller than e^{-180} that was previously (and imprecisely) rounded to 0.0 by NCBI BLAST results in an incorrect 0.0 value. In this less than 0.1% occurrences of an extremely small but non-zero e-value, iBLAST ensures that this imprecise rounding does *not* occur.

Merging two search results with correct e-value statistics

The hits reported by both searches are statistically significant. Once we correct e-values for both the current search result and the new search result, we merge the hits into a single sorted list. Because iBLAST reports some *better* scoring hits that NCBI BLAST misses (explained in detail in Section "iBLAST finds better scoring hits that are missed by NCBI BLAST"), reporting only max_target_seqs hits will result in missing some of the lower-scoring hits from NCBI BLAST. So, we store and report $2 \times max\_target\_seqs$ hits. Algorithm 1 documents the procedure to merge the hits from two results for the same query. All statistical parameters dependent on total database size is re-calibrated to recompute or re-scale the e-values. The hits are selected in the ascending order of their e-values (descending order of their scores).

Algorithm 1 Merging results for Karlin-Altschul/Spouge statistics

Input: $result1, result2$
$merged\_result \leftarrow \Phi$
recompute/re-scale e-values
$m, n \leftarrow 0$
for $i = 1 \rightarrow num\_of\_hits$ **do**
    $e\text{-}value1, score1 \leftarrow min(result1.alignment[m].hsps)$
    $e\text{-}value2, score2 \leftarrow min(result2.alignment[n].hsps)$
    if ($e\text{-}value1 < e\text{-}value2$) or ($e\text{-}value1 == e\text{-}value2$ and $score1 > score2$) **then**
        $merged\_result.add(result1.alignments[m])$
        increment m
    else
        $merged\_result.add(result2.alignments[n])$
        increment n
    end if
end for
return $merged\_result$

Additional details on recomputing and re-scaling e-values is provided in Section "e-value correction."

iBLAST implementation

We develop iBLAST for performing BLAST search as an extension to the NCBI BLAST code. It consists of Python wrapper scripts around the extended BLAST code and uses NCBI BLAST programs as black-box routines. Fig. 4 shows the software stack of iBLAST, which consists of three major components: (1) user interface, (2) incremental logic, and (3) record database. These modules interact with BLAST databases through the BLAST+ programs.

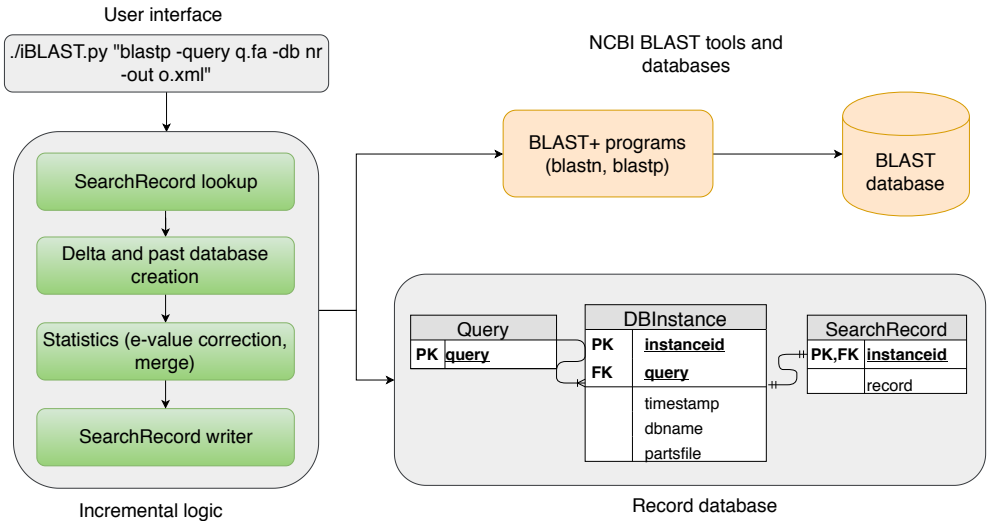


Fig 4. Software stack of iBLAST. The user can initiate a search using the user interface. The search parameters are then passed to the "Incremental logic" module. After performing an incremental search, this module's back-end corrects the e-value statistics and merges the result. The "Incremental logic" module looks into an external lightweight database module called the (*Record database*) to decide whether and how to perform the incremental search. For the actual search and delta database creation, we use NCBI BLAST tools such as blastdbcmd, blastdbalias, blastp, and blastn.

Command-line user interface. In our current version, we provide a command-line user interface for iBLAST, which provides NCBI BLAST-like search options.

Incremental logic. This module decides whether to perform a new BLAST search based on current results. Whenever the user requests a new BLAST search, this module checks for any pre-existing search result. If it does not find any pre-existing result, it performs a regular NCBI BLAST; but if there is a pre-existing result, the module first compares the database instance from the time of the past search with that of the present search. If there is any difference in the database size, this module builds a delta

database that consists of the difference in these two instances. For computing the delta database, this module compares the lists of filenames of the two most recent instances and constructs a database alias using the difference (see Section "Computing delta database" for details). It then performs a new BLAST search only against the delta database and merges the previous result with the new incremental result after statistical correction for e-values. This module allows multiple updates to current searches with little extra time investment. The "Incremental logic" module contains four sub-modules: (1) SearchRecord lookup, (2) Delta and past database creation, (3) Statistics, and (4) SearchRecord writer.

1. ***SearchRecord lookup.*** This sub-module looks for an existing search result with the help of the record database.

2. ***Delta and past database creation.*** This sub-module constructs a delta database by comparing the current database against the database's past instance and performs a BLAST search on the incremental database.

3. ***Statistics.*** This sub-module reads the past and the new incremental search results; it then re-evaluates the e-values in both results and merges them according to their recomputed/re-scaled e-values.

4. ***SearchRecord writer.*** This module writes the updated search result in one of the NCBI BLAST formats.

Whenever the user initiates a BLAST job, the above "Incremental Logic" module first checks if a current search result is available. A delta database consisting of the newly added sequences is constructed if there is a search result against an outdated BLAST database. A BLAST search is then performed against the delta database (i.e., incremental database). In the final stage, the current search result and the incremental search result are merged, and the associated e-values are corrected.

Record database for storing incremental search results Whenever the user performs a BLAST search, iBLAST saves meta-information (e.g., the size of the database and a list of the filenames) about the instance of the database and the search result in a lightweight SQLite database. We design iBLAST to save a minimalist index

structure and size information that requires only a few bytes of storage. We keep the search parameters along with the search results as well. iBLAST does *not* save any redundant copy of any part of the actual sequence database. It stores only the most recent result for a specific query and a database, which keeps the storage overhead to a minimum.

Case studies

To demonstrate the efficiency and benefits of using the iBLAST program over standard NCBI BLAST, we analyze different scenarios on actual nucleotide and protein sequence datasets as case studies.

Case study I: method verification

We explore the scenario where hits from a collection of 100 query sequences are updated to account for the growth of NCBI sequence databases across the duration of the project. To demonstrate the application's use for BLAST programs that use Karlin-Altschul statistics, we ran blastn against a nucleotide database (growing subsets of NCBI nt) for 100 nucleotide sequences from *Bombus impatiens* available at `ftp://ftp.ncbi.nlm.nih.gov/genomes/Bombus_impatiens/CHR_Un/bim_ref_BIMP_2.1_chrUn.fa.gz`. To demonstrate its utility on BLAST programs that use Spouge statistics, we ran blastp against a non-redundant protein database (a growing subset of NCBI nr) for 100 protein sequences from *Bombus impatiens* assembly available at `ftp://ftp.ncbi.nlm.nih.gov/genomes/Bombus_impatiens/protein/protein.fa.gz`.

We demonstrate iBLAST's fidelity and performance over three time periods for case study I, as shown in Fig. 5(A). The instances for nucleotide database changes through time as follows:

- **Time 0:** The nucleotide database comprises 44.5% of the fully available nt database. Both NCBI BLAST and iBLAST search on the same database.
- **Time 1:** The nucleotide database comprises 62.7% of the nt database. While NCBI BLAST searches 62.7% of nt, iBLAST searches only 18.2% of nt. The database grew by 40.8% ($= (62.7 - 44.5)/44.5$) from time 0.

- **Time 2:** The nucleotide database comprises 84.1% of the nt database. While NCBI BLAST searches 84.1% of nt, iBLAST searches only 21.4% of nt. The database grew by 34.1% (= (84.1 − 62.7)/62.7) from time 1.

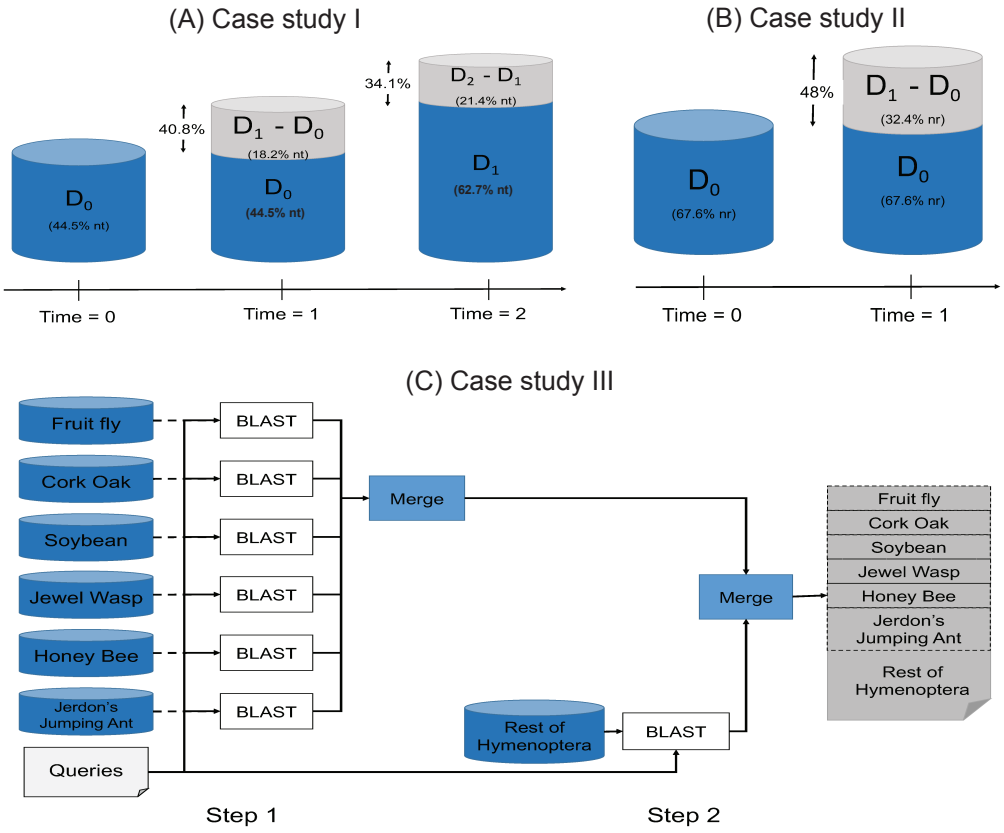


Fig 5. Experimental design of three case studies. (A) Case study I: Incremental addition of sequences in the nt database over three time periods. (B) Case study II: Incremental addition of sequences in the nr database over two time periods. (C) Case study III: Incremental search of taxon-specific databases.

For this case study, we also similarly applied NCBI BLAST and iBLAST to an evolving nr database. That is, the instances of the protein database change over time (in a similar way to the nt database, as captured by Fig. 5(A)). Specifically, the nr database comprises 35.4%, 47.5%, and 60.0% of nr at times 0, 1, and 2, respectively. The protein database grew by 34.1% (i.e., (47.5 − 35.4)/35.4) and 26.3% (i.e., (67.5 − 35.4)/48) by times 1 and 2, respectively, from the earlier time periods. Table 1 provides detailed information about the evolving protein database instances as well as the e-value and hit performance of NCBI BLAST and iBLAST, respectively.

Additional details on the creation of the incremental database can be found in Section "Creating experimental databases."

Case study II: updating a query re-annotation of a novel transcriptomics dataset

Our second case study mimics a typical scenario in a transcriptome re-annotation project where a transcriptome is BLAST-ed after a certain period of time as a part of a re-annotation pipeline. This case study uses a novel dataset not yet available on the NCBI BLAST database — a *de novo* assembled transcriptome of the venom gland of an Oak gall wasp (see below) — and thus, the identity of the assembled sequence was unknown, and the sequence was not available to BLAST to itself.

As shown in Fig. 5(B), we conduct a BLAST search for the same query set for database instances at two time instances:

- **Time 0:** The database comprises 67.6% of the non-redundant database nr (nr accessed on August 2018). Both tools perform the search on this same 67.6% of the database.
- **Time 1:** The database comprises 100% of the non-redundant database nr. While NCBI BLAST performs a search on 100% of nr, iBLAST only needs to search 32.4% of nr as it can reuse the search results from time 0. iBLAST merges the result from time 0 with the incremental search result after e-value correction.

We constructed these two database instances by combining database parts using the blastdb_aliastool utility packaged with BLAST+.

Given the number of queries from the *de novo* assembled transcriptome, it would take a few *months* to complete the search on a single processor core. We ran this experiment with 640 cores distributed across 20 compute nodes (where each node contained dual 16-core Intel Xeon processors, i.e., E5-2683 v4), partitioning the $17,927$ queries into 20 query files and assigning each file per node. Given that each node would run a subset of queries against the same database, there is no need to recompute the statistics for these results before we merge them.

Distributing workload across nodes. The workload across all the nodes should be relatively balanced so that computation for each of the 20 query files finishes roughly simultaneously. To ensure such load balancing, we partitioned the queries using the following strategy. We randomize the order of the queries and partition them so that each partition has roughly the same number of residues. We compare this strategy against the straightforward approach of partitioning the queries in a linear order by putting roughly the same number of queries in each partition.

Case study III: Taxon-based incremental approach

Our third case study presents a special case of using a taxon-based incremental approach to obtain a fast, cost-effective, and biologically relevant results for sequence similarity. To achieve this goal, we examine the genes contained within an assembled transcriptome of the venom gland of a gall wasp of oak trees, the hedgehog gall wasp (***Acraspis erinacei***), a taxon lacking a closely related species with a genome in the nr database. Gall wasps are a group of parasitic wasps that inject their eggs into plant tissues, and through processes yet unknown, they induce changes in plant development at the injection site. These changes result in constructing a niche for the gall wasp by inducing predictable modifications of plant tissues that both protect the wasp from the environment and feed the developing wasp. Genes important for inducing changes in the plant's development are thought to be produced in the female venom gland during oviposition [19]. We performed separate BLAST searches of the hedgehog gall wasp venom gland against transcriptomes of the closest relatives to gall wasps with curated genomes including three fairly equidistant taxa [20] — the parasitic wasp ***Nasonia vitripennis***, the honey bee ***Apis mellifera***, and the ant ***Harpegnathos saltator***, — as well as the more distant model insect, ***Drosophila melanogaster***, upon which many insect gene annotations are based. We also performed BLAST searches against the transcripts of an oak tree, ***Quercus suber***, to determine if some genes belonged to the host as well as a model plant, the soybean, ***Glycine max***.

Using iBLAST, we performed a blastp search individually against each of the databases and merged the results using the statistics module from iBLAST. After this initial search, we then added to this analysis all remaining Hymenopteran species using iBLAST to assess the impact of adding more taxa on the top BLAST hits and further

demonstrate the potential of iBLAST to add taxa progressively, as shown in Fig. 5(C). We performed a blastp search against those seven subsets' merged database to determine whether the same hits would have been found from our concatenated incremental analysis as from a combined single-instance run. These results were further compared with blastp results obtained by searching the complete nr database, allowing us to determine how well we captured the full dataset with this taxon sub-sampling approach.

Data collection for case studies II and III

To obtain the venom gland transcriptome, 15 venom glands were dissected from newly emerged adult females from wild-collected oak (oak spp.) hedgehog galls *Acraspis erinacei* and placed in RNAlater stabilization solution. Pooled tissues were homogenized in lysis buffer using a Bead Ruptor 12 (Omni International) with additional lysis with a 26-gauge syringe. RNA was extracted from the sample using the RNaqueous Micro kit followed by DNase I treatment as specified by the kit and confirmed to be of good quality using the Bioanalyzer 2100 (Agilent). The Illumina HiSeq library was prepared from 200 ng RNA using the TruSeq Stranded mRNA kit and sequenced in 150 bp single-end reads across two Rapid Run lanes on the Illumina HiSeq 2500 (Penn State Genomics Core Facility, University Park, USA) along with nine other barcoded wasp samples.

Raw sequence data (30596191 reads, available at NCBI SRA achieve under Accession ID XXXXXXXXX) quality was assessed using FastQC v0.10.0 (available at `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`), and appropriate trimming for Illumina single-end reads was conducted using Trimmomatic v0.35 [21] to remove adapters and low quality bases (ILLUMINACLIP:TruSeq3-SE:2:30:10 LEADING:20 TRAILING:20 MINLEN:50 AVGQUAL:20), which removed 0.12% of the total reads and *de novo* transcriptome assembly from these QC-passed trimmed reads($30,559,248$ in total) was performed on the Trinity RNA-Seq *de novo* Assembler (version: trinityrnaseq r20140717) [22]. The transcriptome assembly consists of $44,440$ transcripts with the contig N50 of 865 bases. Transdecoder v5.3.0 (available at `https://github.com/TransDecoder/TransDecoder/wiki`) was used to predict $17,927$ protein sequences which were used as queries for Case studies II and III.

Discussion

In this paper, we have introduced iBLAST, an intelligent and incremental local-alignment tool that delivers identical results to NCBI BLAST (along with additional *better* hits missed by NCBI BLAST) and does so with much better performance than NCBI BLAST. The development of novel statistical methods of e-value correction enables our approach to combine multiple search results performed at different times or with different groups (e.g., taxa). Our statistical correction facilitates novel ways of performing sequence alignment tasks and incorporating domain knowledge. For a δ fraction increase in the database size, iBLAST can perform $(1 + \delta)/\delta$ times faster than NCBI BLAST (i.e., 10% growth in database size will yield an 11-fold speedup for iBLAST over NCBI BLAST). We should note that for a small increase in the database size (which is the most likely scenario between two searches), iBLAST delivers a large speedup factor. Furthermore, iBLAST discovers better hits than NCBI BLAST. While iBLAST finds 100% of the hits that NCBI BLAST reports in the same order, iBLAST also reports many additional high-scoring hits that NCBI BLAST misses due to an early approximation used by the heuristic search algorithm in NCBI BLAST.

With the expansion of genetic (sequencing) data available in NCBI, the computational time for large-scale analyses becomes increasingly burdensome, resulting in analyses that take months to complete with a substantial cost, both financially and with respect to "time to solution." This problem is aggravated by cheaper sequencing technology leading to ever-larger genome assembly/transcriptomics projects with substantially more samples to analyze. Our iBLAST tool can help relieve this cost burden. Utilization of iBLAST can enable frequent iterative updates for re-annotation of genome and transcriptome assemblies at a much lower cost (with respect to computational time and financial cost), which is useful given the rapid changes in the nr databases across the duration of a project or its aftermaths. We can add specific datasets of interest to previous searches, such as scenarios involving the availability of new genome releases or conducting large phylogenetic studies. As demonstrated in the final case study, we can use the program in transcriptomic or metagenomics projects by merging the results of knowledge-guided BLAST searches only on biologically relevant groups. The approach used in that case study enables iterative exploration by taxon

and facilitates BLAST results' curation.

Our iBLAST software can work as a wrapper around other fast BLAST implementations and provide multiplicative speedup on the wrapped applications' speedup. A similar approach can benefit other sequence similarity tools and their various implementations if the statistics for correcting the respective statistical significance values (analog to e-value) of the results are available. We aim to develop a standard pipeline for other popular sequence-similarity search tools to combine results through a framework for automated statistical correction in future work. Through its statistical correction formulas and software stack, iBLAST presents the potential to make other sequence similarity-search tools faster by utilizing past search results and incorporating domain knowledge in a period when sequence database is growing exponentially.

Acknowledgements

We like to thank Andy Deans and István Mikó for their contributions to the data collection and Istvan Albert and Ramu Anandakrishnan for providing valuable feedback on earlier versions of the manuscript. We also thank Yang Pu and Jingwei Zhang for working on the initial model of iBLAST concerning Karlin-Altschul statistics.

This work was supported in part by the Institute for Critical Technology and Applied Science (ICTAS), an institute dedicated to transformative, interdisciplinary research for a sustainable future (`http://www.ictas.vt.edu`). Computation for this research were performed on Virginia Tech's *Advanced Research Computing (VT ARC)* and Pennsylvania State University's *Institute for CyberScience Advanced CyberInfrastructure (ICS-ACI)*. Contributions from Sarthok Rasique Rahman were supported by NSF DEB #1453473.

References

1. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. Journal of molecular biology. 1990;215(3):403–410.
2. Benson DA, Cavanaugh M, Clark K, Karsch-Mizrachi I, Ostell J, Pruitt KD, et al. GenBank. Nucleic Acids Research. 2017;46(D1):D41–D47. doi:10.1093/nar/gkx1094.
3. Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, et al. Big data: astronomical or genomical? PLoS biology. 2015;13(7):e1002195.
4. Eddy SR. Profile hidden Markov models. Bioinformatics (Oxford, England). 1998;14(9):755–763.
5. Buchfink B, Xie C, Huson DH. Fast and sensitive protein alignment using DIAMOND. Nature methods. 2014;12(1):59.
6. Loh PR, Baym M, Berger B. Compressive genomics. Nature biotechnology. 2012;30(7):627.
7. Zhang J, Wang H, Lin H, Feng Wc. cuBLASTP: Fine-grained parallelization of protein sequence search on a GPU. In: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. IEEE; 2014. p. 251–260.
8. Zhang J, Misra S, Wang H, Feng Wc. muBLASTP: database-indexed protein sequence search on multicore CPUs. BMC bioinformatics. 2016;17(1):443.
9. Darling AE, Carey L, Feng WC. The design, implementation, and evaluation of mpiBLAST. Los Alamos National Laboratory; 2003.
10. de Castro MR, dos Santos Tostes C, Dávila AM, Senger H, da Silva FA. SparkBLAST: scalable BLAST processing using in-memory operations. BMC bioinformatics. 2017;18(1):318.
11. Youssef K, Feng Wc. SparkLeBLAST: Scalable Parallelization of BLAST Sequence Alignment Using Spark. In: 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid). Melbourne, Victoria, Australia; 2020.

12. Haft DH, DiCuccio M, Badretdin A, Brover V, Chetvernin V, O'Neill K, et al. RefSeq: an update on prokaryotic genome annotation and curation. Nucleic acids research. 2017;46(D1):D851–D860.

13. O'Leary NA, Wright MW, Brister JR, Ciufo S, Haddad D, McVeigh R, et al. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. Nucleic acids research. 2015;44(D1):D733–D745.

14. Shah N, Nute MG, Warnow T, Pop M. Misunderstood parameter of NCBI BLAST impacts the correctness of bioinformatics workflows. Bioinformatics. 2018; p. bty833. doi:10.1093/bioinformatics/bty833.

15. González-Pech RA, Stephens TG, Chan CX. Commonly misunderstood parameters of NCBI BLAST and important considerations for users. Bioinformatics. 2018;doi:10.1093/bioinformatics/bty1018.

16. NCBI. The Statistics of Sequence Similarity Scores; 2019. `https://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html`.

17. Park Y, Sheetlin S, Ma N, Madden TL, Spouge JL. New finite-size correction for local alignment score distributions. BMC research notes. 2012;5(1):286.

18. Lagnel J, Tsigenopoulos CS, Iliopoulos I. NOBLAST and JAMBLAST: New Options for BLAST and a Java Application Manager for BLAST results. Bioinformatics. 2009;25(6):824–826.

19. Vårdal H. Venom gland and reservoir morphology in cynipoid wasps. Arthropod structure & development. 2006;35(2):127–136.

20. Peters RS, Krogmann L, Mayer C, Donath A, Gunkel S, Meusemann K, et al. Evolutionary history of the Hymenoptera. Current Biology. 2017;27(7):1013–1018.

21. Bolger AM, Lohse M, Usadel B. Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics. 2014;30(15):2114–2120.

22. Grabherr MG, Haas BJ, Yassour M, Levin JZ, Thompson DA, Amit I, et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. Nature biotechnology. 2011;29(7):644.

Supporting information

Growth of Sequence Data

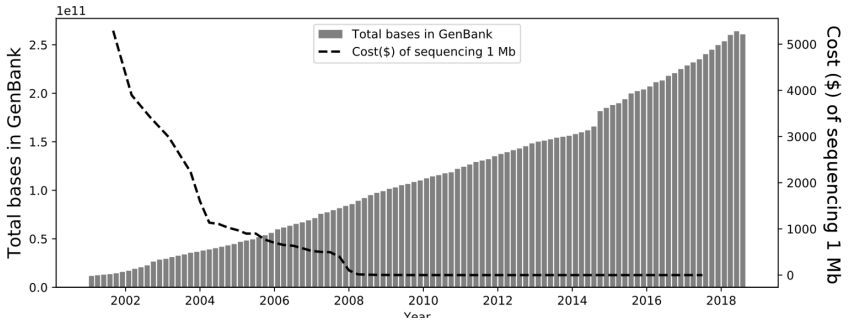


Fig S1. Growth of sequence database size compared to the sequencing cost. Increasing GenBank database size available at `https://www.ncbi.nlm.nih.gov/genbank/statistics/`, accessed on September 15, 2018) follows a decreasing trend in sequencing cost (available at `https://github.com/TransDecoder/TransDecoder/wiki`, accessed on September 15, 2018) .

Existing e-value correction software and their features

mpiBLAST

mpiBLAST [9] is a parallel implementation of NCBI BLAST on the cluster. It segments the database, ports the segments into different nodes of a cluster, and runs parallel BLAST search jobs against database segments on different nodes. Once the parallel search jobs return, it aggregates the search result. It has two important contributions. First, it achieves super-linear speedup by reducing IO overhead (time spent in reading and writing hard-disk storage). Second, it is the first parallel BLAST tool to provide exact e-value statistics in contrast to approximate e-value statistics of other contemporary parallel implementations of NCBI BLAST.

mpiBLAST's exact e-value statistics requires two steps. First, it collects the necessary statistical parameters for the entire database by performing a pseudo-run of the BLAST engine against the global database. Once it has the global parameters, it passes the global parameters (such as whole database length n, the total number of sequences N) to the parallel search jobs against segmented databases. mpiBLAST modifies some functionalities of NCBI BLAST (blast.c, blastdef.h, blastkar.c, and blastutl.c) so that global parameters can be fed externally and that information can be

used to calculate exact e-values.

For accurate e-value correction, mpiBLAST requires prior knowledge of the entire database.

NOBLAST

NOBLAST [18] provides new options for NCBI BLAST. It offers a way to correct e-values when split databases are used and the results need to be aggregated. E-value computation requires knowledge about the entire database size, the number of sequences in the whole database N and the total length of the database n. Using the values N, n and Karlin-Altschul statistical parameters which are independent of database size, the e-value can be computed using Karlin-Altschul statistics. First, NOBLAST computes the length adjustment using the knowledge about the complete original database, then, it computes effective search space using length adjustment, and finally, it computes the e-value using effective search space.

In principle, NOBLAST takes a similar approach to mpiBLAST, as both provide global statistical parameters to the search jobs against a segmented database so that that exact e-value can be computed. While mpiBLAST's main contribution is a parallel implementation and e-value correction comes from the need of producing the same output as the sequential counterpart, NOBLAST's main contribution is an e-value correction. Both tools require prior knowledge about the entire database. Both tools were developed before Spouge's e-value statistics were introduced, so they didn't address e-value corrections for the BLAST programs that use Spouge's statistics.

e-value correction

We use algorithm 2 to recompute e-values for BLAST programs using Karlin-Altschul statistics. We first aggregate database sizes from two input results and use the aggregated size$N$ to compute length adjustment l. Using N and l, we recompute e-values for both results.

Algorithm 2 Recomputing e-values for Karlin-Altschul Statistics

Input: $result1, result2$

$n \leftarrow result1.n + result2.n$

$m \leftarrow result1.m$

$N \leftarrow result1.N + result2.N$

$l \leftarrow recompute\_length\_adjustment(n, m, N)$

$recompute\_evalues(result1, l, N)$

$recompute\_evalues(result2, l, N)$

We use algorithm 3 to re-scale e-values for BLAST programs using Spouge statistics. First we aggregate the database sizes for two input results, and scale the e-values by a factor of the ratio between aggregated database size and the individual database size.

Algorithm 3 Re-scale e-values for Spouge statistics

Input: $result1, result2$

$db\_length1 \leftarrow result1.db\_length$

$db\_length2 \leftarrow result2.db\_length$

$db\_length \leftarrow db\_length1 + db\_length2$

$re\text{-}scale\_evalues(result1, \frac{db\_length}{db\_length1})$

$re\text{-}scale\_evalues(result2, \frac{db\_length}{db\_length2})$

Creating experimental databases

Pre-formatted BLAST databases such *nt* and *nr* come in incremental parts. With progression of time, new sequences are packaged in parts and added to the databases.

Databases for case study I

For case study I, we consider three time steps when the nt and nr databases had 30, 40, and 50 parts. For these three time periods, we construct three databases as instances of nt and nr by combining 30, 40, and 50 parts using BLAST tool *blastdb_aliastool*. The incremental databases between two periods are also constructed. Table S1 shows different instances of *nt* databases in three different periods.

Table S2 shows different instances of *nr* databases in three different periods.

Table S1. Incremental nt Databases for case study I.

Time Period	Database parts	Number of sequences	Number of bases	Longest sequence length (bases)
T_0	0-29	25,117,275	80,740,533,243	774,434,471
$T_0 \rightarrow T_1$	30-39	8,389,596	33,008,962,097	275,920,749
T_1	0-39	33,506,871	113,749,495,340	774,434,471
$T_1 \rightarrow T_2$	40-59	8,891,258	38,722,333,261	129,927,919
T_2	0-49	42,398,129	152,471,828,601	774,434,471

Table S2. Incremental nr databases for case study I

Time Period	Database parts	Number of sequences	Number of residues	Longest sequence length (residues)
T_0	0-29	49,468,463	17,686,779,866	36,507
$T_0 \rightarrow T_1$	30-39	15,878,318	6,065,300,773	35,523
T_1	0-39	65,346,781	23,752,080,639	36,507
$T_1 \rightarrow T_2$	40-49	16,448,075	6,278,067,810	38,105
T_2	0-49	81,794,856	30,030,148,449	38,105

Databases for case study II

We construct nr database instances for time 0 and 1 by combining 64 and 90 parts respectively. We combine these parts using *blastdb_aliastool*.

Load-balancing via query partitioning

For case study II and III, we have partitioned 17927 queries into 20 query files based on number of residues after randomizing the order instead of a more straightforward

Table S3. Incremental nr databases for case study II

Time Period	Database	Number of sequences	Total residues	Longest sequence length (residues)
T_0	0-63	109,407,071	40,077,622,077	38,105
$T_0 \rightarrow T_1$	64-90	52,860,187	19,192,851,238	74,488
T_1	0-90	162,267,258	59,270,473,315	74,488

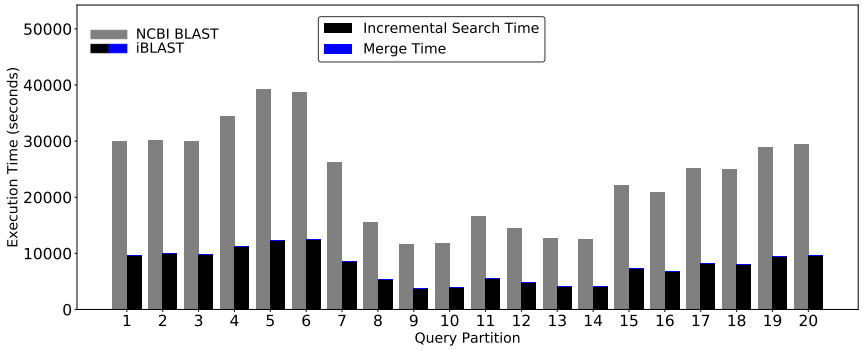


Fig S2. Load imbalance resulted from a naive query partitioning. Execution time when a straightforward query partitioning scheme is adopted, which results in significant lack of load balancing. The standard deviation for the execution times for both incremental and NCBI BLAST searches are large (2748 and 8727 seconds respectively).

partitioning based on number of queries while keeping the original order.

If we partition the queries by making sure each partition has roughly same number of queries without disrupting their order, we get a range of execution times demonstrating lack of proper load balancing. The standard deviation in iBLAST search times is 2748 seconds and standard deviation in NCBI BLAST search times is 8727 seconds. This means the compute nodes have to wait idly for $1 - 3$ hours on average. Fig S2 demonstrates the lack of load balancing.

In contrast, when we first randomize the order of the queries and then partition the queries by making sure that all partitions have the roughly same number of residues, the standard deviations fall to 150 and 487 seconds respectively (Fig S3).

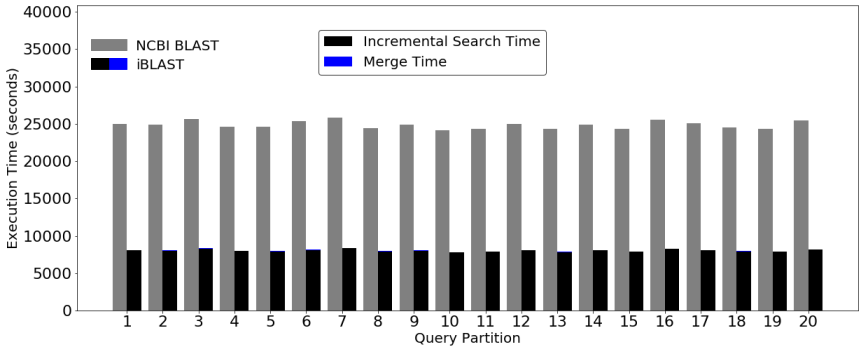


Fig S3. Load balance demonstrated by our proposed strategy. Execution time when our improvised query partitioning scheme is adopted which results in better load balancing. The standard deviation for the execution times for both incremental and NCBI BLAST searches are minimal compared to the naive strategy (150 and 487 seconds respectively).

Explanation for NCBI BLAST missing many top hits

Due to the early cutoff of max target sequence used by its heuristic algorithm. NCBI BLAST performs search in two phases. In earlier phase (ungapped extension), it starts with matching a seed sub-string between target and query sequence and then extends the matching pair in both direction without allowing any gap. In this phase, BLAST algorithm assigns some scores to these matching pairs and keeps only the very high scoring pairs using a cutoff determined by e-value cutoff or number of maximum hits. In the gapped phase, these selected high scoring pairs are further extended in both directions while allowing gaps and these evolved pairs get changed scores. Some of the pairs that did not make the cut during the ungapped extension, can become high scoring pairs. For a larger database, these missed opportunities are higher in number because there are more potential pairs in the ungapped phase. Since iBLAST is combining results from smaller databases, it misses relatively smaller number of those high scoring hits compared to NCBI BLAST.

Computing delta database

A formatted NCBI database consists of several index files and actual sequences are partitioned into several 1GB partitions. For example non redundant protein database *nr* has 100 1GB partitions named as nr.00, nr.01, . . . , nr.99. While saving an instance of the database, AdaBLAST saves a list of these file names. Let, at time t_1, database *db*

had P_{t_1} parts, so the *Record Database* will save the instance by saving the list $L_1 = db.00, db.01, \ldots, db.(P_{t_1} - 1)$. At time t_2, nr got updated and now it has P_{t_2} data files. The instance of db at time t_2 is the list $L_2 = db.00, db.01, \ldots, db.(P_{t_2} - 1)$. To compute delta database between times t_1, t_2, the *Incremental Logic* module will first compute the difference between the two lists saved as the two instances. The difference is $\delta L = L_2 - L_1 = db.(P_{t_1}), \ldots, db.(P_{t_2} - 1)$. We then use *blastdb_aliastool* distributed with command-line BLAST to make the delta database using the files in δL.

There is one caveat, the last file ($db.(P_{t_1} - 1)$) in the instance at t_1 might have been updated with new sequences and by not including them, we might miss some of the potential hits. To mitigate this effect, we include $db.(Pt_1 - 1)$ in the delta database. When recomputing or re-scaling the e-values, we use the correct previous and current database sizes (Fig S4).

It takes few milliseconds to compute the delta database from two instances.

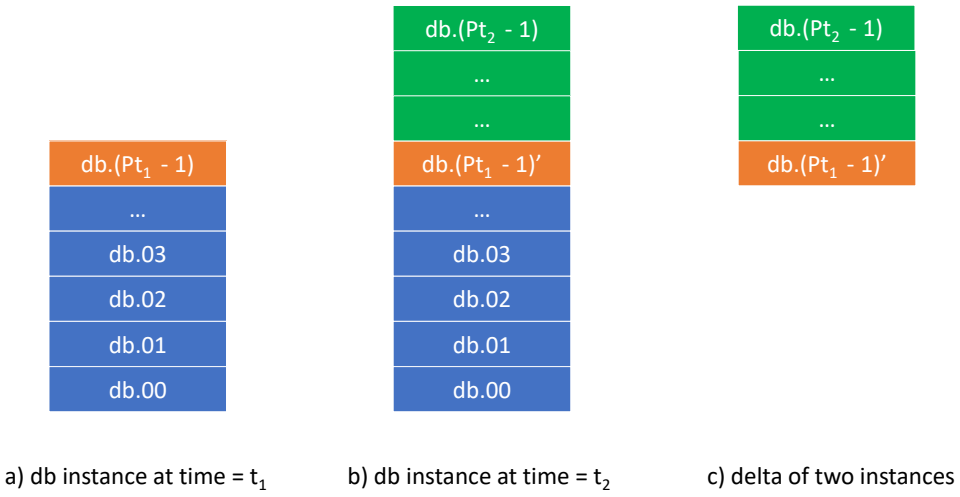


Fig S4. Delta database computation from two instances of the same database *db*. At time t_1, the instance has Pt_1 parts. At time t_2, the instance has Pt_2 parts. In both of these instances, the part $db.(Pt_1)$ is common. However this part is potentially updated. So, we take all the parts from $db.(Pt_1)$ to $db.(Pt_2)$ to compute the delta database.