cblaster

Release 1.0.5

Jun 22, 2020

Contents:

CHAPTER 1

Usage

There are two search modes, specified by the *–mode* argument. By default (i.e. *–mode* not specified), it will be set to *remote*, and a fully remote *cblaster* search will begin using NCBI's BLAST API. Alternatively, *local* mode performs a search against a local *diamond* database, which is much quicker (albeit requiring some initial setup).

1.1 Performing a remote search via the NCBI BLAST API

At a minimum, a search could look like one of the following:

```
$ cblaster search -qf query.fasta
$ cblaster search -qi QBE85649.1 QBE85648.1 QBE85647.1 QBE85646.1 ...
```

This will launch a remote search against the non-redundant (nr) protein database, retrieve and parse the results, then report any blocks of hits to the terminal. By default, hits are only reported if they are above 30% percent identity and 50% query coverage, and have an e-value below 0.01. If we wanted to be stricter, we could change those values with the following:

```
$ cblaster search -qf query.fasta --min_identity 70 --min_coverage 90 --evalue 0.001
```

You can also pass in NCBI search queries using *-eq / –entrez_query* to pre-filter the target database, which can result in vastly reduced run-times and more targeted results. For example, to only search against *Aspergillus* sequences:

```
$ cblaster search -qf query.fasta --entrez_query "Aspergillus"[ORGN]
```

Look here for a full description of Entrez search terms.

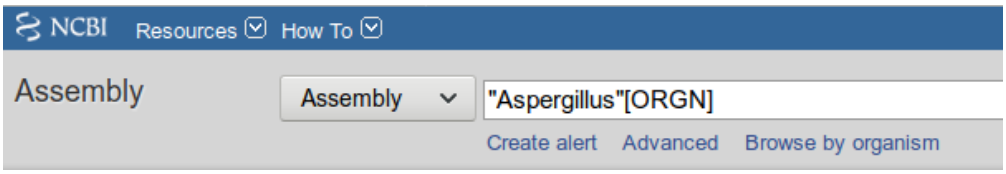
1.2 Searching a local database using DIAMOND

Alternatively, a local DIAMOND database can be searched by specifying:

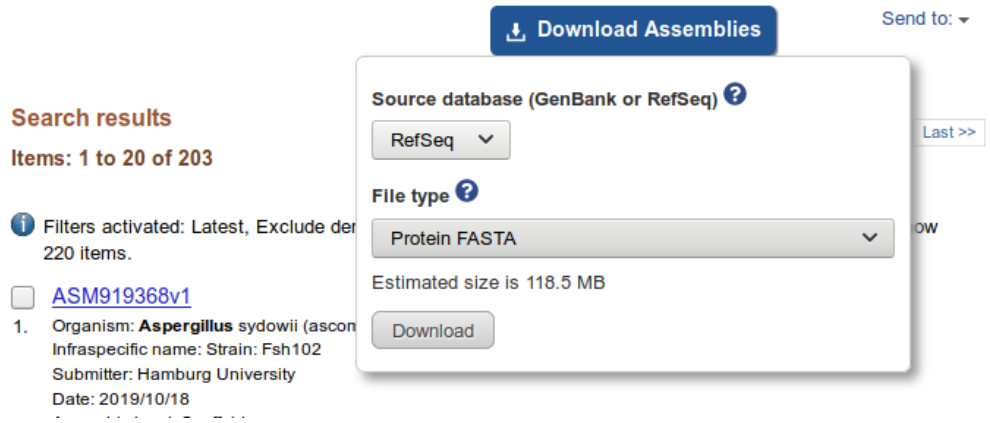
```
$ cblaster search -qf query.fasta --mode local --database db.dmnd
```

For this to work, the database must consist of sequences derived from NCBI, such that their identifiers can be used for retrieval of sequences/genomic context. The easiest way to set this up is via NCBI's batch assembly download option. For example, to build a database of *Aspergillus* protein sequences:

1. Search the NCBI Assembly database for *Aspergillus* genomes



2. Click 'Download Assemblies', select 'Protein FASTA' and click 'Download'



3. Extract all FASTA files and concatenate them

```
$ pigz -d *.gz
$ cat *.faa >> proteins.faa
```

4. Build the DIAMOND database

```
$ diamond makedb --in proteins.faa --db proteins
...
$ ls
database.faa
database.dmnd
```

5. Run *cblaster* against the newly created databse

```
$ cblaster search -m local -qf query.fa -db database.dmnd <options>
```

Alternatively, you could use ncbi-genome-download to retrieve the sequences from the command line.

1.3 Performing a fully local search

cblaster can also perform fully local searches, forgoing the need for any interaction with NCBI whatsoever. To do this, *cblaster* builds a faux-database JSON file from a list of GenBank files, and generates a *diamond* database of all protein

sequences in this database. Then, *cblaster* searches can be run against the created *diamond* database, and genomic context obtained from the JSON file. For example:

1. Build *diamond* and JSON databases

```
$ cblaster makedb path/to/folder/*.gbk mydatabase
[12:06:32] INFO - Parsing 3 files...
[12:06:32] INFO - 1. /path/to/folder/one.gbk
[12:06:33] INFO - 2. /path/to/folder/two.gbk
[12:06:34] INFO - 3. /path/to/folder/three.gbk
[12:07:21] INFO - Writing FASTA file with database sequences: mydatabase.faa
[12:07:21] INFO - Building DIAMOND database: mydatabase.dmnd
[12:07:36] INFO - Building JSON database: mydatabase.json
```

Now, everything should be in the folder:

```
$ ls
mydatabase.faa
mydatabase.dmnd
mydatabase.json
```

Note that although this produces a FASTA file with all protein sequences in the database, the file is NOT actually required for searches. It is merely created for the purpose of building the *diamond* database, and can easily be recreated by loading the JSON database in code as a *database.DB* instance and running *write_fasta()* on an open file handle.

2. Run a *cblaster* search against the built databases

```
$ cblaster search -qf query.fasta --json mydatabase.json -db mydatabase.db <options>
```

CHAPTER 2

Classes

This module stores the classes (Organism, Scaffold, Hit) used in cblaster.

class cblaster.classes.**Hit**(*query*, *subject*, *identity*, *coverage*, *evalue*, *bitscore*)

A BLAST hit identified during a cblaster search.

This class is first instantiated when parsing BLAST results, and is then updated with genomic coordinates after querying either the Identical Protein Groups (IPG) resource on NCBI, or a local JSON database.

query

Name of query sequence.

Type str

subject

Name of subject sequence.

Type str

identity

Percentage identity (%) of hit.

Type float

coverage

Query coverage (%) of hit.

Type float

evalue

E-value of hit.

Type float

bitscore

Bitscore of hit.

Type float

start

Start of subject sequence on corresponding scaffold.

Type int

end
End of subject sequence on corresponding scaffold

Type int

strand
Orientation of subject sequence ('+' or '-').

Type str

copy(***kwargs*)
Creates a copy of this Hit with any additional args.

classmethod from_dict(*d*)
Loads class from dict.

to_dict()
Serialises class to dict.

values(*decimals=4*)
Formats hit attributes for printing.

Parameters **decimals** (*int*) – Total decimal places to show in score values.

Returns List of formatted attribute strings.

class cblaster.classes.**Organism**(*name*, *strain*, *scaffolds=None*)
A unique organism containing hits found in a cblaster search.

Every strain (or lack thereof) is a unique Organism, and will be reported separately in cblaster results.

name
Organism name, typically the genus and species epithet.

Type str

strain
Strain name of this organism, e.g. CBS 536.65.

Type str

scaffolds
Scaffold objects belonging to this organism.

Type dict

classmethod from_dict(*d*)
Loads class from dict.

full_name
The full name (including strain) of the organism. Note: if strain found in name, returns just name.

to_dict()
Serialises class to dict.

total_hit_clusters
Counts total amount of hit clusters in this Organism.

class cblaster.classes.**Scaffold**(*accession*, *clusters=None*, *subjects=None*)
A genomic scaffold containing hits found in a cblaster search.

accession
Name of this scaffold, typically NCBI accession.

Type str

hits
Hit objects located on this scaffold.

Type list

clusters
Clusters of hits identified on this scaffold.

Type list

classmethod from_dict(*d*)
Loads class from dict.

to_dict()
Serialises class to dict.

class cblaster.classes.**Serializer**
JSON serialisation mixin class.

Classes that inherit from this class should implement *to_dict* and *from_dict* methods.

classmethod from_dict(*d*)
Loads class from dict.

classmethod from_json(*js*)
Instantiates class from JSON handle.

to_dict()
Serialises class to dict.

to_json(*fp=None*, ***kwargs*)
Serialises class to JSON.

class cblaster.classes.**Session**(*queries*, *params*, *organisms=None*)
Stores the state of a cblaster search.

This class stores query proteins, search parameters, Organism objects created during searches, as well as methods for generating summary tables. It can also be dumped to/loaded from JSON for re-filtering, plotting, etc.

```
>>> s = Session()
>>> with open("session.json", "w") as fp:
...     s.to_json(fp)
>>> with open("session.json") as fp:
...     s2 = Session.from_json(fp)
>>> s == s2
True
```

queries
Names of query sequences.

Type list

params
Search parameters.

Type dict

organisms
Organism objects created in a search.

Type list

format (*form*, *fp=None*, ***kwargs*)
Generates a summary table.

Parameters

- **form** (*str*) – Type of table to generate ('summary' or 'binary').
- **fp** (*file handle*) – File handle to write to.
- **human** (*bool*) – Use human-readable format.
- **headers** (*bool*) – Show table headers.

Raises ValueError – *form* not 'binary' or 'summary'

Returns Summary table.

classmethod from_dict(*d*)
Loads class from dict.

to_dict()
Serialises class to dict.

class cblaster.classes.**Subject**(*hits=None*, *ipg=None*, *start=None*, *end=None*, *strand=None*)
A sequence representing one or more BLAST hits.

This class is instantiated during the contextual lookup stage. It is important since it allows for subject sequences which hit >1 of the query sequences, while still staying non-redundant.

hits
Hit objects referencing this subject sequence.

Type list

ipg
NCBI Identical Protein Group (IPG) id.

Type int

start
Start of sequence on parent scaffold.

Type int

end
End of sequence on parent scaffold.

Type int

strand
Strandedness of the sequence ('+' or '-').

Type str

classmethod from_dict(*d*)
Loads class from dict.

to_dict()
Serialises class to dict.

CHAPTER 3

Context

CHAPTER 4

Database

CHAPTER 5

Helpers

cblaster.helpers.**efetch_sequences**(*headers*)
Retrieve protein sequences from NCBI for supplied accessions.

This function uses EFetch from the NCBI E-utilities to retrieve the sequences for all synthases specified in *headers*. It then calls *fasta.parse* to parse the returned response; note that extra processing has to occur because the returned FASTA will contain a full sequence description in the header line after the accession.

Parameters **headers** (*list*) – Valid NCBI sequence identifiers (accession, GI, etc.).

cblaster.helpers.**efetch_sequences_request**(*headers*)
Launch E-Fetch request for a list of sequence accessions.

Parameters **headers** (*list*) – NCBI sequence accessions.

Raises requests.HTTPError – Received bad status code from NCBI.

Returns Response returned by requests library.

Return type requests.models.Response

cblaster.helpers.**form_command**(*parameters*)
Flatten a dictionary to create a command list for use in subprocess.run()

cblaster.helpers.**get_program_path**(*aliases*)
Get programs path given a list of program names.

Parameters **aliases** (*list*) – Program aliases, e.g. ["diamond", "diamond-aligner"]

Raises ValueError – Could not find any of the given aliases on system $PATH.

Returns Path to program executable.

cblaster.helpers.**get_sequences**(*query_file=None*, *query_ids=None*)
Convenience function to get dictionary of query sequences from file or IDs.

Parameters

- **query_file** (*str*) – Path to FASTA file containing query protein sequences.
- **query_ids** (*list*) – NCBI sequence accessions.

Raises `ValueError` – Did not receive values for query_file or query_ids.

Returns Dictionary of query sequences keyed on accession.

Return type sequences (dict)

`cblaster.helpers.`**`parse_fasta`**(*handle*)
Parse sequences in a FASTA file.

Returns Sequences in FASTA file keyed on their headers (i.e. > line)

CHAPTER 6

Local

cblaster.local.**diamond**(*fasta*, *database*, *max_evalue=0.01*, *min_identity=30*, *min_coverage=50*, *cpus=1*)

Launch a local DIAMOND search against a database.

Parameters

- **fasta** (*str*) – Path to FASTA format query file
- **database** (*str*) – Path to DIAMOND database generated with cblaster makedb
- **max_evalue** (*float*) – Maximum e-value threshold
- **min_identity** (*float*) – Minimum identity (%) cutoff
- **min_coverage** (*float*) – Minimum coverage (%) cutoff
- **cpus** (*int*) – Number of CPU threads for DIAMOND to use

Returns Rows from DIAMOND search result table (split by newline)

Return type list

cblaster.local.**parse**(*results*, *min_identity=30*, *min_coverage=50*, *max_evalue=0.01*)

Parse a string containing results of a BLAST/DIAMOND search.

Parameters

- **results** (*list*) – Results returned by diamond() or blastp()
- **min_identity** (*float*) – Minimum identity (%) cutoff
- **min_coverage** (*float*) – Minimum coverage (%) cutoff
- **max_evalue** (*float*) – Maximum e-value threshold

Returns Hit objects representing hits that surpass scoring thresholds

Return type list

cblaster.local.**search**(*database*, *query_file=None*, *query_ids=None*, ***kwargs*)

Launch a new BLAST search using either DIAMOND or command-line BLASTp (remote).

Parameters

- **database** (*str*) – Path to DIAMOND database
- **query_file** (*str*) – Path to FASTA file containing query sequences
- **query_ids** (*list*) – NCBI sequence accessions

Raises `ValueError` – No value given for query_file or query_ids

Returns Parsed rows with hits from DIAMOND results table

Return type list

CHAPTER 7

Main

CHAPTER 8

Remote

This module handles all interaction with NCBI's BLAST API, including launching new remote searches, polling for completion status, and retrieval of results.

cblaster.remote.**check**(*rid*)

Check completion status of a BLAST search given a Request Identifier (RID).

Parameters **rid** (*str*) – NCBI BLAST search request identifier (RID)

Returns Search has completed successfully and hits were reported

Return type bool

Raises

- ValueError – Search has failed. This is caused either by program error (in which case, NCBI requests you submit an error report with the RID) or expiration of the RID (only stored for 24 hours).
- ValueError – Search has completed successfully, but no hits were reported.

cblaster.remote.**parse**(*handle*, *query_file=None*, *query_ids=None*, *max_evalue=0.01*, *min_identity=30*, *min_coverage=50*)

Parse Tabular results from remote BLAST search performed via API.

Since the API provides no option for returning query coverage, which is a metric we want to use for filtering hits, query sequences must be passed to this function so that their lengths can be compared to the alignment length.

Parameters

- **handle** (*list*) – File handle (or file handle-like) object corresponding to BLAST results. Note that this function expects an iterable of tab-delimited lines and performs no validation/error checking
- **query_file** (*str*) – Path to FASTA format query file
- **query_ids** (*list*) – NCBI sequence identifiers
- **max_evalue** (*float*) – Maximum e-value

- **min_identity** (*float*) – Minimum percent identity
- **min_coverage** (*float*) – Minimum percent query coverage

Returns Hit objects corresponding to criteria passing BLAST hits

Return type list

cblaster.remote.**poll**(*rid*, *delay=60*, *max_retries=-1*)

Poll BLAST API with given Request Identifier (RID) until results are returned.

As per NCBI usage guidelines, this function will only poll once per minute; this is calculated each time such that wait is constant (i.e. accounts for differing response time on the status check).

Parameters

- **rid** (*str*) – NCBI BLAST search request identifier (RID)
- **delay** (*int*) – Total delay (seconds) between polling
- **max_retries** (*int*) – Maximum number of polling attempts (-1 for unlimited)

Returns BLAST search results split by newline

Return type list

cblaster.remote.**retrieve**(*rid*, *hitlist_size=500*)

Retrieve BLAST results corresponding to a given Request Identifier (RID).

Parameters

- **rid** (*str*) – NCBI BLAST search request identifiers (RID)
- **hitlist_size** (*int*) – Total number of hits to retrieve

Returns BLAST search results split by newline, with HTML parts removed

Return type list

cblaster.remote.**search**(*rid=None*, *query_file=None*, *query_ids=None*, *min_identity=0.3*, *min_coverage=0.5*, *max_evalue=0.01*, ***kwargs*)

Perform a remote BLAST search via the NCBI's BLAST API.

This function launches a new search given a query FASTA file or list of valid NCBI identifiers, polls the API to check the completion status of the search, then retrieves and parses the results.

It is also possible to call other BLAST variants using the program argument.

Parameters

- **rid** (*str*) – NCBI BLAST search request identifier (RID)
- **query_file** (*str*) – Path to FASTA format query file
- **query_ids** (*list*) – NCBI sequence identifiers
- **min_identity** (*float*) – Minimum percent identity
- **min_coverage** (*float*) – Minimum percent query coverage
- **max_evalue** (*float*) – Maximum e-value

Returns Hit objects corresponding to criteria passing BLAST hits

Return type list

cblaster.remote.**start**(*query_file=None*, *query_ids=None*, *database='nr'*, *program='blastp'*, *megablast=False*, *filtering='F'*, *evalue=10*, *nucl_reward=None*, *nucl_penalty=None*, *gap_costs='11 1'*, *matrix='BLOSUM62'*, *hitlist_size=500*, *threshold=11*, *word_size=6*, *comp_based_stats=2*, *entrez_query=None*)

Launch a remote BLAST search using NCBI BLAST API.

Note that the HITLIST_SIZE, ALIGNMENTS and DESCRIPTIONS parameters must all be set together in order to mimic max_target_seqs behaviour.

Usage guidelines: 1. Don't contact server more than once every 10 seconds 2. Don't poll for a single RID more than once a minute 3. Use URL parameter email/tool 4. Run scripts weekends or 9pm-5am Eastern time on weekdays if >50 searches

For a full description of the parameters, see:

1. *BLAST API documentation<https://ncbi.github.io/blast-cloud/dev/api.html>*
2. *BLAST documentation <https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Bl*

Parameters

- **query_file** (*str*) – Path to a query FASTA file
- **query_ids** (*list*) – Collection of NCBI sequence identifiers
- **database** (*str*) – Target NCBI BLAST database
- **program** (*str*) – BLAST variant to run
- **megablast** (*bool*) – Enable megaBLAST option (only with BLASTn)
- **filtering** (*str*) – Low complexity filtering
- **evalue** (*float*) – E-value cutoff
- **nucl_reward** (*int*) – Reward for matching bases (only with BLASTN/megaBLAST)
- **nucl_penalty** (*int*) – Penalty for mismatched bases (only with BLASTN/megaBLAST)
- **gap_costs** (*str*) – Gap existence and extension costs
- **matrix** (*str*) – Scoring matrix name
- **hitlist_size** (*int*) – Number of database sequences to keep
- **threshold** (*int*) – Neighbouring score for initial words
- **word_size** (*int*) – Size of word for initial matches
- **comp_based_stats** (*int*) – Composition based statistics algorithm
- **entrez_query** (*str*) – NCBI Entrez search term for pre-filtering the BLAST database

Returns Request Identifier (RID) assigned to the search rtoe (int): Request Time Of Execution (RTOE), estimated run time of the search

Return type rid (str)

CHAPTER 9

Formatters

cblaster result formatters.

cblaster.formatters.**add_field_whitespace**(*rows*, *lengths*)
Fills table fields with whitespace to specified lengths.

cblaster.formatters.**binary**(*session*, *hide_headers=False*, *delimiter=None*, *key=<built-in function len>*, *attr='identity'*, *decimals=4*)
Generates a binary summary table from a Session object.

cblaster.formatters.**generate_header_string**(*text*, *symbol='-'*)
Generates a 2-line header string with underlined text.

```
>>> header = generate_header_string("header string", symbol="*")
>>> print(header)
header string
*************
```

cblaster.formatters.**get_cell_values**(*queries*, *subjects*, *key=<built-in function len>*, *attr=None*)
Generates the values of cells in the binary matrix.

This function calls some specified key function (def. max) against all values of a specified attribute (def. None) from Hits inside Subjects which match each query. By default, this function will just count all matching Hits (i.e. len() is called on all Hits whose query attribute matches). To find maximum identities, for example, provide key=max and attr='identity' to this function.

Parameters

- **queries** (*list*) – Names of query sequences.
- **subjects** (*list*) – Subject objects to generate vlaues for.
- **key** (*callable*) – Some callable that takes a list and produces a value.
- **attr** (*str*) – A Hit attribute to calculate values with in key function.

cblaster.formatters.**get_maximum_row_lengths**(*rows*)
Finds the longest lengths of fields per column in a collection of rows.

cblaster.formatters.**humanise**(*rows*)
Formats a collection of fields as human-readable.

cblaster.formatters.**summarise_subjects**(*subjects*, *decimals=4*, *hide_headers=True*, *delimiter=None*)
Generates a summary table for a hit cluster.

Parameters

- **hits** (*list*) – collection of Hit objects
- **decimals** (*int*) – number of decimal points to show
- **show_headers** (*bool*) – show column headers in output
- **human** (*bool*) – use human-readable format

Returns summary table

CHAPTER 10

Parsers

Argument parsers.

CHAPTER 11

Plot

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

P

Q

R

S

T

V