The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

16: BLAST API

Thomas Madden

Jason Papadopoulos

Christiam Camacho

George Coulouris

Kevin Bealer

madden@ncbi.nlm.nih.gov
papadopo@ncbi.nlm.nih.gov
camacho@ncbi.nlm.nih.gov
coulouri@ncbi.nlm.nih.gov
bealer@ncbi.nlm.nih.gov

Created: August 22, 2006.
Last Update: April 13, 2010.

Overview

The overview for this chapter consists of the following topics:

- Introduction
- Chapter Outline

Introduction

BLAST (Basic Local Alignment Search Tool) is used to perform sequence similarity searches. Most often this means that BLAST is used to search a sequence (either DNA or protein) against a database of other sequences (either all nucleotide or all protein) in order to identify similar sequences. BLAST has many different flavors and can not only search DNA against DNA or protein against protein but also can translate a nucleotide query and search it against a protein database as well as the other way around. It can also compute a "profile" for the query sequence and use that for further searches as well as search the query against a database of profiles. BLAST is available as a web service at the NCBI, as a stand-alone binary, and is built into other tools. It is an extremely versatile program and probably the most heavily used similarity search program in the world. BLAST runs on a multitude of different platforms that include Windows, MacOS, LINUX, and many flavors of UNIX. It is also under continuing development with new algorithmic innovations. Multiple references to BLAST can be found at http://www.ncbi.nlm.nih.gov/BLAST/blast_references.shtml.

The version of BLAST in the NCBI C++ Toolkit was rewritten from scratch based upon the version in the C Toolkit that was originally introduced in 1997. A decision was made to break the code for the new version of BLAST into two different categories. There is the "core" code of BLAST that is written in vanilla C and does not use any part of the NCBI C or C++ Toolkits. There is also the "API" code that is written in C++ and takes full advantage of the tools provided by the NCBI C++ Toolkit. The reason to write the core part of the code in vanilla C was so that the same code could be used in the C Toolkit (to replace the 1997 version) as well as to make it possible for researchers interested in algorithmic development to work with the core of BLAST independently of any Toolkit. Even though the core part was written without the benefit of the C++ or C Toolkits an effort was made to conform to the Programming Policies and Guidelines chapter of this book. Doxygen-style comments are used to allow API documentation to be automatically generated (see the BLAST Doxygen link at http://www.ncbi.nlm.nih.gov/IEB/

The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

ToolBox/CPP_DOC/doxyhtml/group__AlgoBlast.html). Both the core and API parts of BLAST can be found under algo/blast in the C++ Toolkit.

An attempt was made to isolate the user of the BLAST API (as exposed in algo/blast/api) from the core of BLAST, so that algorithmic enhancements or refactoring of that code would be transparent to the API programmer as far as that is possible. Since BLAST is continually under development and many of the developments involve new features it is not always possible or desirable to isolate the API programmer from these changes. This chapter will focus on the API for the C++ Toolkit. A few different search classes will be discussed. These include the CLocalBlast class, typically used for searching a query (or queries) against a BLAST database; CRemoteBlast, used for sending searches to the NCBI servers; as well as CBl2Seq, useful for searching target sequences that have not been formatted as a BLAST database.

Chapter Outline

CLocalBlast

The class CLocalBlast can be used for searches that run locally on a machine (as opposed to sending the request over the network to use the CPU of another machine) and search a query (or queries) against a preformatted BLAST database, which holds the target sequence data in a format optimal for BLAST searches. The demonstration program blast_demo.cpp illustrates the use of CLocalBlast. There are a few different CLocalBlast constructors, but they always take three arguments reflecting the need for a query sequence, a set of BLAST options, and a set of target sequences (e.g., BLAST database). First we discuss how to construct these arguments and then we discuss how to access the results.

The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

Query Sequence

The classes that perform BLAST searches expect to be given query sequences in one of a few formats. Each is a container for one or more query sequences expressed as CSeq_loc objects, along with ancillary information. In this document we will only discuss classes that take either a SSeqLoc or a TSeqLocVector, which is just a collection of SSeqLoc's.

CBlastInput is a class that converts an abstract source of sequence data into a format suitable for use by the BLAST search classes. This class may produce either a TSeqLocVector container or a CBlastQueryVector container to represent query sequences. As mentioned above we limit our discussion to the TSeqLocVector class here.

CBlastInput can produce a single container that includes all the query sequences, or can output a batch of sequences at a time (the combined length of the sequences within each batch can be specified) until all of the sequences within the data source have been consumed.

Sources of sequence data are represented by a CBlastInputSource, or a class derived from it. CBlastInput uses these classes to read one sequence at a time from the data source and convert to a container suitable for use by the BLAST search classes.

An example use of CBlastInputSource is CBlastFastaInputSource, which represents a stream containing fasta-formatted biological sequences. Usually this class represents a collection of sequences residing in a text file. One sequence at a time is read from the file and converted into a BLAST input container.

CBlastFastaInputSource uses CBlastInputConfig to provide more control over the file reading process. For example, the read process can be limited to a range of each sequence, or sequence letters that appear in lowercase can be scheduled for masking by BLAST. CBlastInputConfig can be used by other classes to provide the same kind of control, although not all class members will be appropriate for every data source.

Options

The BLAST options classes were designed to allow a programmer to easily set the options to values appropriate to common tasks, but then modify individual options as needed. Table 1 lists the supported tasks.

The CBlastOptionsFactory class offers a single static method to create CBlastOptionsHandle subclasses so that options applicable to all variants of BLAST can be inspected or modified. The actual type of the CBlastOptionsHandle returned by the Create() method is determined by its EProgram argument (see Table 1). The return value of this function is guaranteed to have reasonable defaults set for the selected task.

The CBlastOptionsHandle class encapsulates options that are common to all variants of BLAST, from which more specific tasks can inherit the common options. The subclasses of CBlastOptionsHandle should present an interface that is more specific, i.e.: only contain options relevant to the task at hand, although it might not be an exhaustive interface for all options available for the task. Please note that the initialization of this class' data members follows the template method design pattern, and this should be followed by subclasses also. Below is an example use of the CBlastOptionsHandle to create a set of options appropriate to "blastn" and then to set the expect value to non-default values:

```
using ncbi::blast;

CRef<CBlastOptionsHandle>
```

```
 opts_handle(CBlastOptionsFactory::Create(eBlastn));
opts_handle->SetEvalueThreshold(1e-10);
blast(query, opts_handle, db);
```

The CBlastOptionsHandle classes offers a Validate() method in its interface which is called by the BLAST search classes prior to performing the actual search, but users of the C++ BLAST options APIs might also want to invoke this method to ensure that any exceptions thrown by the BLAST search classes do not originate from an incorrect setting of BLAST options. Please note that the Validate() method throws a CBlastException in case of failure.

If the same type of search (e.g., nucleotide query vs. nucleotide database) will always be performed, then it may be preferable to create an instance of the derived classes of the CBlastOptionsHandle. These classes expose an interface that is relevant to the task at hand, but the popular options can be modified as necessary:

```
using ncbi::blast;

CRef<CBlastNucleotideOptionsHandle> nucl_handle(new
CBlastNucleotideOptionsHandle);
...
nucl_handle->SetTraditionalBlastnDefaults();
nucl_handle->SetStrandOption(objects::eNa_strand_plus);
...
CRef<CBlastOptionsHandle> opts = CRef<CBlastOptionsHandle> (&*nucl_handle);
CLocalBlast blast(query_factory, opts, db);
```

The CBlastOptionsHandle design arranges the BLAST options in a hierarchy. For example all searches that involve protein-protein comparisons (including proteins translated from a nucleotide sequence) are handled by CBlastProteinOptionsHandle or a subclass (e.g., CBlastxOptionsHandle). A limitation of this design is that the introduction of new algorithms or new options that only apply to some programs may violate the class hierarchy. To allow advanced users to overcome this limitation the GetOptions() and SetOptions() methods of the CBlastOptionsHandle hierarchy allow access to the CBlastOptions class, the lowest level class in the C++ BLAST options API which contains all options available to all variants of the BLAST algorithm. No guarantees about the validity of the options are made if this interface is used, therefore invoking Validate() is *strongly* recommended.

Target Sequences

One may specify a BLAST database to search with the CSearchDatabase class. Normally it is only necessary to provide a string for the database name and state whether it is a nucleotide or protein database. It is also possible to specify an entrez query or a vector of GI's that will be used to limit the search.

Results

The Run() method of CLocalBlast returns a CSearchResultSet that may be used to obtain results of the search. The CSearchResultSet class is a random access container of CSearchResults objects, one for each query submitted in the search. The CSearchResult class provides access to alignment (as a CSeq_align_set), the query Cseq_id, warning or error messages that were generated during the run, as well as the filtered query regions (assuming query filtering was set).

The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

CRemoteBlast

The CRemoteBlast class sends a BLAST request to the SPLITD system at the NCBI. This can be advantageous in many situations. There is no need to download the (possibly) large BLAST databases to the user's machine; the search may be spread across many machines by the SPLITD system at the NCBI, making it very fast; and the results will be kept on the NCBI server for 36 hours in case the users wishes to retrieve them again the next day. On the other hand the user must select one of the BLAST databases maintained by the NCBI since it is not possible to upload a custom database for searching. Here we discuss a CRemoteBlast constructor that takes three arguments, reflecting the need for a query sequence(s), a set of BLAST options, and a BLAST database. Readers are advised to read the CLocalBlast section before they read this section.

Query Sequence

A TSeqLocVector should be used as input to CRemoteBlast. Please see the section on CLocalBlast for details.

Options

CBlastOptionsFactory::Create() can again be used to create options for CRemoteBlast. In this case though it is necessary to set the second (default) argument of Create() to CBlastOptions::eRemote.

Target Sequences

One may use the CSearchDatabase class to specify a BLAST database, similar to the method outlined in the CLocalBlast section. In this case it is important to remember though that the user must select from the BLAST databases available on the NCBI Web site and not one built locally.

Results

After construction of the CRemoteBlast object the user should call one of the SubmitSync() methods. After this returns the method GetResultSet() will return a CSearchResultSet which the user can interrogate using the same methods as in CLocalBlast. Additionally the user may obtain the request identifier (RID) issued by the SPLITD system with the method GetRID().

Finally CRemoteBlast provides a constructor that takes a string, which it expects to be an RID issued by the SPLITD system. This RID might have been obtained by an earlier run of CRemoteBlast or it could be one that was obtained from the NCBI SPLITD system via the web page. Note that the SPLITD system will keep results on it's server for 36 hours, so the RID cannot be older than that.

The Uniform Interface

The ISeqSearch class is an abstract interface class. Concrete subclasses can run either local (CLocalSeqSearch) or remote searches (CRemoteSeqSearch). The concrete classes will only perform an intersection of the tasks that CLocalBlast and CRemoteBlast can perform. As an example, there is no method to retrieve a Request identifier (RID) from subclasses of ISeqSearch as this is supported only for remote searches but not for local searches. The methods supported by the concrete subclasses and the return values are similar to those of CLocalBlast and CRemoteBlast.

The NCBI C++ Toolkit Book

The NCBI C++ Toolkit Book

CBl2Seq

CBl2Seq is a class useful for searching a query (or queries) against one or more target sequences that have not been formatted as a BLAST database. These sequences may, for example, come from a user who pasted them into a web page or be fetched from the Entrez or ID1 services at the NCBI. The CBl2Seq constructors all take three arguments, reflecting the need for a set of query sequences, a set of target sequences, and some information about the BLAST options or program type to use. In this section it is assumed the reader has already read the previous section on CLocalBlast.

The BLAST database holds the target sequence data in a format optimal for BLAST searches, so that if a target sequence is to be searched more than a few times it is best to convert it to a BLAST database and use CLocalBlast.

Query Sequence

The query sequence (or sequences) is represented either as a SSeqLoc (for a single query sequence) or as a TSeqLocVector (in the case of multiple query sequences). The CBlastInput class, described in the CLocalBlast section, can be used to produce a TSeqLocVector.

Options and Program Type

The CBl2Seq constructor takes either an EProgram enum (see Table 1) or CBlastOptionsHandle (see relevant section under CLocalBlast). In the former case the default set of options for the given EProgram are used. In the latter case it is possible for the user to set options to non-default values.

Target Sequences

The target sequence(s) is represented either as a SSeqLoc or TSeqLocVector.

Results

The Run() method of the CBl2Seq class returns a collection of CSeq_align_set's. The method GetMessages() may be used to obtain any error or warning messages generated during the search.

Sample Applications

The following are sample applications that demonstrate the usage of the CBl2Seq and CLocalBlast classes respectively:

- blast_sample.cpp
- blast_demo.cpp

Table 1: List of tasks supported by the CBlastOptionsHandle. “Translated nucleotide” means that the input was nucleotide, but the comparison is based upon the protein. PSSM is a “position-specific scoring matrix”. The “EProgram” can be used as an argument to CBlastOptionsFactory::Create

EProgram (enum)	**Default Word-size**	**Query type**	**Target type**	**Notes**
eBlastN	11	Nucleotide	Nucleotide	
eMegablast	28	Nucleotide	Nucleotide	Optimized for speed and closely related sequences
eDiscMegablast	11	Nucleotide	Nucleotide	Optimized for cross-species matches
eBlastp	3	Protein	Protein	
eBlastx	3	Translated nucleotide	Protein	
eTblastn	3	Protein	Translated nucleotide	
eTblastx	3	Translated nucleotide	Translated nucleotide	
eRPSBlast	3	Protein	PSSM	Can very quickly identify domains
eRPSTblastn	3	Translated nucleotide	PSSM	
ePSIBlast	3	PSSM	Protein	Extremely sensitive method to find distant homologies
ePHIBlastp	3	Protein	Protein	Uses pattern in query to start alignments