Tampere University

Katariina Latvala

DEVELOPMENT OF A BLAST-BASED COMMAND LINE TOOL FOR SEQUENCE ALIGNMENT

Faculty of Information
Technology and Communication
Sciences
Bachelor's thesis
May 2019

ABSTRACT

Katariina Latvala: Development of a BLAST-based command line tool for sequence alignment
Bachelor's thesis
Tampere University
Bachelor of Science – Information Technology
May 2019

This thesis work studies the development of a BLAST (Basic Local Alignment Search Tool) - based command line tool for the use of biological sequence alignment. The focus of the work is in identifying the most important software quality criteria based on the target user base, and how to produce such a program using existing resources as efficiently as possible.

The first part of the work covers the basic principles of biological sequence alignment, the BLAST algorithm and related existing software applications. The next part introduces software quality criteria and typical models for their application. A brief literature review of the state of software development and quality in the field of bioinformatics is included to illustrate the most common problems that one might need to navigate in the development of a program such as the one discussed in this thesis. The second part of the thesis discusses the design of the program, selected quality criteria as well as technical aspects of the development.

The results section details the results of the project with an overview of the program architecture. Following this is a discussion on how well the implementation filled the desired software quality criteria and ideas for further development.

Keywords: BLAST, bioinformatics, software development, software quality

PREFACE

I would like to thank Thomas Kaminski for bringing me the idea of developing together a BLAST-based computational tool to help their laboratory with their research. The program, LBlastPyCli developed in the summer of 2018 is the basis for the program discussed in this thesis and has already spanned another exciting future project.

Tampere, 28 April 2019

Katariina Latvala

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

BLAST	Basic Local Alignment Search Tool
DNA	Deoxyribonucleic acid
RNA	Ribonucleic acid
NCBI	National Center for Bioinformatics Technology
NCBI BLAST	NCBI's BLAST-based software product family
HSP	High-scoring Segment Pair
FASTA	widely used text file format for biological sequence data
ISO	International Organization for Standardization
CISQ	Consortium for IT Software Quality
LBLASTCLI	Local BLAST Command Line Interface

1. INTRODUCTION

In recent decades technological advancements in gene sequencing has allowed us to read the entire genomes of several species from worms to humans. Sequencing of genes as well as their functional products in the form of RNA and proteins form a major part of genetics research. All this sequential information of nucleotides or amino acids must be analysed and interpreted somehow, with comparison to identified sequences being a common analysis method. This comparison, or more specifically alignment, of a sequence of interest to identified ones and the calculation of a similarity score between them can be done by several algorithms, of which BLAST (Basic Local Alignment Search Tool) [1] is the most popular one. There exist vast public databases of identified and annotated sequences as well as tools such as the NCBI's (National Center for Biotechnology Information) BLAST web application that allow researchers to perform these types of analyses. However, there are many cases where this is not feasible; the NCBI's public servers limit traffic so one cannot use large input files (holding several query sequences), and in some cases researchers are limited by propriety or otherwise non-published data. Cloud computing is an option, but an expensive one. Very few research grants are geared singularly towards computation or bioinformatics, and research groups rarely allocate sufficient funds for such within their budgets [2].

Due to the previously mentioned reasons, being able to run BLAST locally is an important task for many researchers. The NCBI has already developed open source tools for this, but most researchers lack the skills – and the time to learn how – to use them. Thus, there is a clear need for a simple and efficient command line tool that will allow the user to run BLAST on local databases and produce output of the best matches, that can then also be used as input for the NCBI's BLAST web service to get more detailed information.

This thesis details the development of such a command line tool, with an emphasis on efficiency and usability, while also conforming with other general software quality criteria.

2. THE COMPUTATIONAL PROBLEM OF GENE SEQUENCE ALIGNMENT

This chapter discusses the theoretical background of gene sequence alignment, the computational problems associated with it and existing computational tools.

2.1 Gene sequences

The genome is the entirety of the genetic material within an organism, and genomics is an important field of research that tries to answer various questions spanning evolution, inheritance and diseases all the way to population genetics and human behaviour. At its core the research relies on the analysis of the primary sequential information encoded within the genome. The building blocks of the genome, deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) consists of long sequences of molecules called nucleotides: adenine [A], cytosine [C], guanine [G] and thymine [T] (replaced by uracil [U] in RNA) [3]. These contain specific regions, genes, that code for proteins, which are the driving force behind all essential physiological processes, and thus their study by the way of identification and analysis is essential. DNA is transcribed into RNA, used as a template to form chains of amino acids, with each nucleotide triplet (codon) encoding for an amino acid and folded into further higher-level structures resulting in a functional protein.

Genes undergo evolution and change the same way that all life does. Nature rarely invents things *de novo,* but rather adapts existing forms and structures. This is fortunate since this similarity or *homology* between a known and a new sequence can be used as a basis to make inferences about the structure and/or function of the new sequence. The identification of sequence homology is usually the first step in discovering the function of a newly sequences gene. [4]

2.2 The basics of sequence alignment

There exists extensive literature on string comparison methods in the field of computer science and some biological sequence alignment methods rely on these very same methods. However, in the biological sphere the concept of *alignment* is an important one; it is not only the overall similarity of compared sequences that is significant, but how the similar or conserved sub-sequences are distributed and the patterns in their contents. [4] Genes evolve and can undergo various mutations in the form of insertions, deletions and substitutions of nucleotides. The frequency of each of these events is non-uniform between nucleotides and species [5], and thus must be considered when trying to align sequences. There are other biological constraints such as the three-dimensional structures of the resultant protein, as dictated by its physicochemical properties, that affect the possible or likely sequences for any given case.

Sequence alignment methods can try to find global (where both sequences are either the same length to start with or gaps in either sequence are allowed to make them uniform in length) or local optimal alignments. Most alignment methods rely on some type of scoring scheme as the basis for finding the optimal alignment between two sequences. The simplest ones assign negative scores for mismatches and positive scores for matches, with the aim of finding the alignment that maximises the overall summed score. More sophisticated scoring systems include various heuristics that account for the biological constraints outlined in the previous paragraph.

Many algorithms for global and local alignment have been developed over the years, but the early dynamic programming algorithms [6][7] are rarely used on their own today, as their computational requirements are impractical when searching through the type of large sequence databases that are common nowadays.

2.3 Basic Local Alignment Search Tool

According to one of its co-developers, Stephen Altschul, BLAST was the first local sequence alignment method to use rigorous statistics as part of its scoring system [8]. If you consider the scenario where you are comparing a 500-nucleotide long query sequence and all its segments to a database of tens of thousands of sequences, the likelihood of getting locally similar segments can be significant, and statistical treatment of possible alignments should form a part of the overall method.

BLAST approximates the results that a dynamic programming algorithm would provide, but at a significantly smaller computational cost [1]. The exact details of the tool are beyond the scope of this work, but the general steps are described below. The query sequence is used to generate a *k*-letter word list. For each of these words, another list of all possible matching words is generated and scored based on some substitution matrix. After this, BLAST picks out only the matches with a score over a certain threshold *T*, saving significantly on computation time compared to methods that examine all the possibilities. The chosen match words are built into a search tree that is then used when scanning the database sequences for exact matches. These high-scoring segment pairs

(HSP) are then extended in both directions as long as the accumulated score (as assigned by the same substitution matrix as in the previous step) stays the same or increases. The same step of filtering out low-scoring cases is repeated, this time with a cut-off score *S*. In the final main step, the chosen HSPs are statistically evaluated to determine whether the score is significant when considering the size of the database (how many sequences the query was compared to), producing a final expectation or *E*-value for each HSP. This *E*-value is most commonly used by researchers and other end users to set a threshold for similarity between the query and the database sequences when using the algorithm or tools based on it.

2.4 Existing computational tools and methods

The paper detailing BLAST was published in 1990 [1] around the time that the internet was taking off and sharing of information like sequence databases was becoming easier. According to Altschul one contributing factor in BLAST's popularity was its integration with the early NCBI sequence and literature databases [8], and later the Entrez Global Query Cross-Database Search System [9] that is a text-based search and retrieval system that allows free access to multiple different research literature and biomolecular databases.

As mentioned in the introduction, the web-based (both browser and API) BLAST application is extensive, but traffic to the NCBI's public servers is limited and can not be used for large automated searches. The open source tools such as the C++ Toolkit [10] and the executable command line tools provide users with many possibilities but require some basic programming knowledge with their installation and application. Even the use of the NCBI BLAST+ executables requires some knowledge of basic scripting and command line tool standards. For a biologist who has only practical experience of using the web-based application and no formal computer science training, learning to use these tools can be judged to be not worth the effort.

The common use case that the NCBI's website is perfectly suited for is when the user has only a single or few unannotated query sequences. They can search and align it with all the annotated sequence databases available or set limits by taxonomy or other features. The general process of such a use case is displayed in Figure 1. The user obtains raw sequence data from some experiment(s) and creates a FASTA file from it (the NCBI BLAST accepts some other formats as well, but FASTA is the standard one). This file is used as query input, as it contains the unannotated sequences that the user wants to identify. The publicly available, annotated sequence databases are used as the target input against which the query input is searched and aligned by the NCBI BLAST application. As output the program generates all the found local alignments, ranked by significance. The user can examine the alignments in detail with interactive graphical tools as well as download the local or full aligned sequences (the matches found in the target input) as FASTA files.

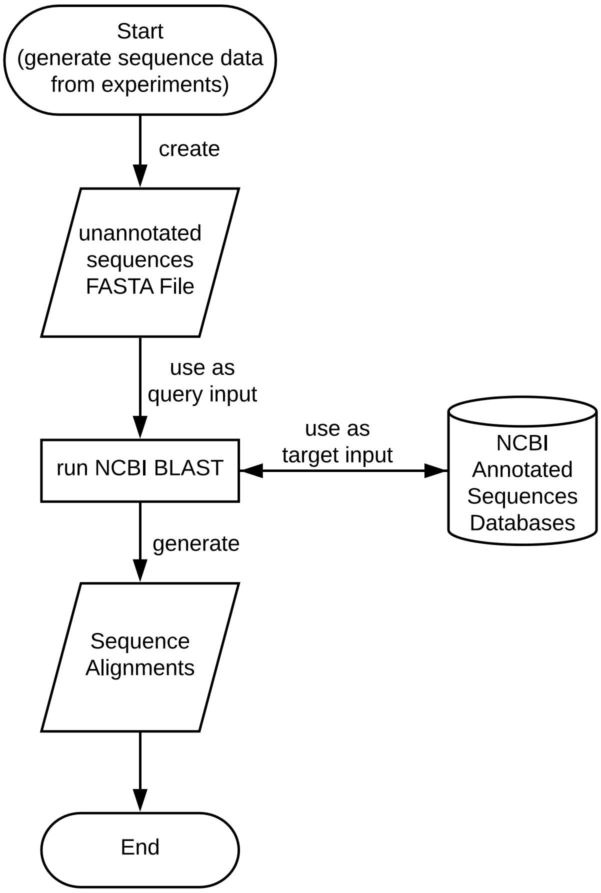


Figure 1. *General use case of NCBI BLAST*

However, the general scenario contrasts with the type of sequence data that many researchers deal with; most of the data might contain already known gene sequences that are not relevant or not match any known sequences. Also, in many cases researchers have hypotheses on what gene sequences they expect to find, based on the sample types, the experimental setup and other associated information. The user has data, a local database if you will, of unannotated sequences that they want to analyse, but the number of sequences is too large to use as input to the NCBI BLAST application to be run on the public servers.

The process can be flipped, and instead of using the experimentally derived unannotated data as query sequences to be searched and aligned against the fully annotated sequences in the public databases as in Figure 1, they can be used as the target input against which some selected annotated sequences are searched and aligned. This flipped use case in described in Figure 2 and is the basis for the program discussed in this thesis. Using the selected annotated sequences as query input and the unannotated sequences as target input, the BLAST algorithm is run in the local environment. As with the use case in Figure 1, the algorithm will output the best local alignments, which can be used to retrieve the full sequences from the target input and written into a FASTA file.

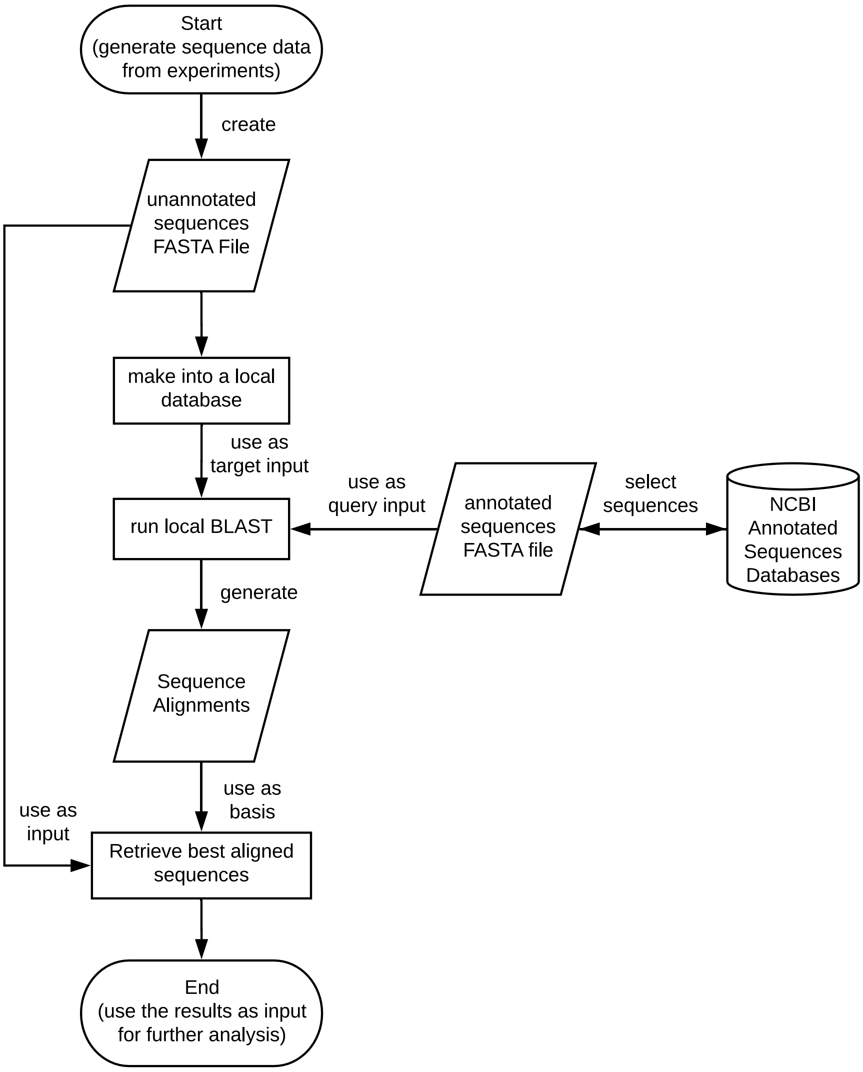

Start
(generate sequence data from experiments)
create
unannotated sequences FASTA File
make into a local database
use as target input
run local BLAST
use as query input
annotated sequences FASTA file
select sequences
NCBI Annotated Sequences Databases
generate
Sequence Alignments
use as basis
use as input
Retrieve best aligned sequences
End
(use the results as input for further analysis)

Figure 2. *Flipped use case of local BLAST*

As was discussed earlier, homology between biological sequences implies a close evolutionary relationship between the organisms or a relationship between the functions that the genes or proteins are involved in. So, for our flipped use case we can use as query sequences either the exact sequences that we're hoping to find (in the case where they are annotated and available) or annotated sequences that are taxonomically closest to the organism our data is of, or the functionally closest. These sequences can be selected from the public databases. The output of the process in Figure 2 can be used as query input for further analysis, including with the NCBI BLAST browser tool or API, as described in the general use case in Figure 1. The difference is that now the query input is much smaller and has already been verified to align with the selected sequences of interest. The NCBI BLAST is only used to verify these alignments, get more detailed information and perhaps get alignments with other homologous sequences.

3. SOFTWARE QUALITY

Quality as a concept is a complicated one. Quantitative measurements of characteristics or entities are a common feature of engineering, but qualitative assessments play an important role as well. This holds true for software quality as well. There are many definitions for the concept of quality in software engineering and software products, as well as standards and frameworks for its assessment.

In the context of software, quality can be divided into two categories; functional quality and structural quality. Functional quality is the fitness for purpose or how well the software meets with specifications or requirements. This is closely tied in with customer or user value. This value can be relative in how the software product or service meets the customer's current needs, anticipates future needs and compares to competitors in the market. On the other hand, it can also be examined in isolation, assessing how well the product meets the requirements of a specific customer group. Thus, functional quality is closely tied in with the specific requirements and user assessment. Structural or non-functional quality deals with the non-functional features or requirements that support functional quality, such as reliability or security.

3.1 Software quality models

Software quality assessment requires models and standards to facilitate communication between different groups and to provide some objective means of measurement. ISO (the International Organization for Standardization) [11] has a Software Product Quality Model as part of its ISO 25000 standards, called the ISO/IEC 25010 [12], shown in Figure 3. This model is comprised of 8 quality characteristics, which can be further divided into sub-characteristics that comprise of attributes that can be verified or measured as part of a quality evaluation system. ISO/IEC 25023:2016 [13], also known as SQuaRE (Systems and software Quality Requirements and Evaluation), is a quantitative evaluation system based on the ISO/IEC 25010.

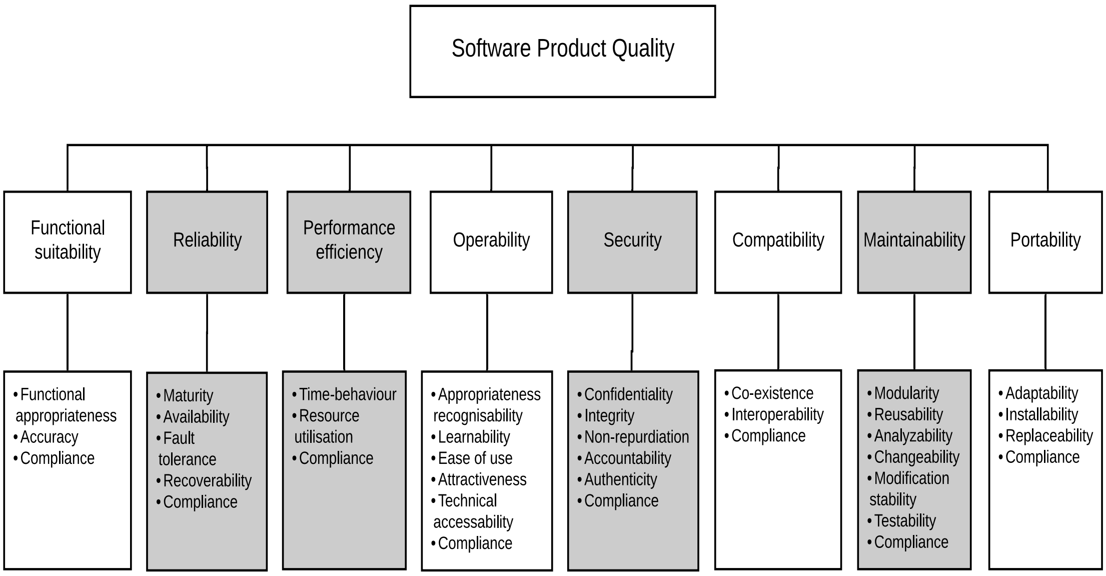


Figure 3. *ISO/IEC 25000 and CISQ (in grey) models. Adapted from* [14]

SQuaRE relies mainly on quality assessment at the behavioural and functional level rather than the code level. Because of this CISQ (Consortium for IT Software Quality) has defined a source code level measures of 4 of the quality characteristics defined in ISO/IEC 25010; reliability, performance efficiency, security and maintainability. These characteristics are highlighted in grey in Figure 3. The CISQ measures cover 86 critical code quality rules that can be detected by automated analysis of the source code at the unit and system level. [14]

3.2 Software quality in bioinformatics

There exists a significant amount of literature on software development and quality in the fields of bioinformatics and the broader field of scientific computation. Material is published by researchers from all fields, including computer scientists and software engineers working on biology-related case studies, and biologists publishing their experimental findings and related computational methods. A full literature review of the topic is outside of the scope of this thesis, but it is possible to draw some information from the literature to guide the design and development process of the program that is the focus of this thesis.

In general there seems to be a distinct lack of software engineering and production standards in the field of bioinformatics due to a general lack of education, training and funding [2][15][16][17]. The curricula of biology-based degrees include limited programming or general computer science skills, and according to one study [15] none of the bioinformatics degrees included compulsory studies in software engineering. Along with education the lack of funding is the other major culprit behind the lacking standards. Only a small fraction of available grants and other funding is geared towards software development or related work [2], and most research groups don’t budget for it. Software is treated as a means to an end and is seen as a less critical than the experimental or theoretical work like hypothesis formulation. However, adherence to good software engineering is critical for the correctness of the results as well as their reproducibility. Testing can be very difficult in a bioinformatics context but is essential for reliability. The program input and output spaces can be vast and complex, making it very difficult to define sufficient testing cases among other problems [18]. For reproducibility, things such as documentation and version control are essential. Most software projects in the field suffer from the “developer as user” problem, which is often accompanied by lacking documentation and version control, informal testing and limited availability of source code (for other potential users such as journal reviewers) [17].

4. THE SOFTWARE PROJECT METHODS AND TOOLS

This chapter focuses on the methods and tools used in the development of the BLAST-based command line tool for sequence alignment, hence forth referred to as LBLASTCLI.

4.1 Quality criteria

Due to the scope of this thesis and the simplicity of the LBLASTCLI program, it was decided that no quantitative quality evaluation systems would be used, but the design and development of the project was guided by several chosen quality criteria.

Firstly, operability in the form of ease of use and learnability. As discussed in previous chapters, the majority of the targeted end users of this program possess minimal IT and programming skills. The general functionality of the program is already available in the form of the various NCBI tools (including the Blast+ executables), but the objective of the project is to make a simple program with maximal automation.

The program should be possible to use with a single line call of options and arguments through the command line or terminal. The program should provide simple instructions on the necessary and allowed input and give sufficient feedback when invalid or otherwise erroneous input is given by the user. The program should conform to common standards for command line tools, such as the ones discussed in [19].

For a software tool used in complex analytical tasks, functional correctness and accuracy is essential. As discussed in subchapter 3.2, the accuracy of the results a program produces is critical for any further research or hypotheses as well as reproducibility. In general, the complexity of the input and output spaces make it difficult for a user to check the correctness of their results. However, in this particular case the users can verify the results independently using the NCBI BLAST tools.

Functional completeness, meaning the extent that the program's functions cover all the specified tasks or user objectives, is another criterion. The program should run BLAST on the user-specified query and target sequences and produce FASTA files of the best aligned sequences as a result. All related sub-tasks, such as database creation from the target input, should be automated.

Performance in terms of time-behaviour and resource utilisation should be maximised. The general use case for any BLAST-based tool can involve very large amounts of data, so time-efficiency is paramount. Also, one of the motivations of this project is the inability of NCBI BLAST to process large inputs through the public servers, and the effort needed for the alternative of manually submitting smaller chunks of the data sequentially.

Resource utilisation in term of memory usage is important due to the potentially very large magnitude of the data. However, performance optimization at the code level should not come at the cost of readability or modularity.

The program should deal with errors accordingly, inform the user and recover independently if possible. The error messages passed to the user should be at the appropriate detail; they should not directly refer to any of the architecture or internal functionality of the program, but rather to the aspects visible at the user interface.

The criteria of modularity, reusability, analysability, modifiability and testability are all universal quality criteria, and not relevant only to this specific program. None of them have a direct effect from the user's perspective, but rather support the other quality characteristics. The program should be modular with as few dependencies as possible, so that each module can be tested and analysed from a performance perspective in isolation. Modularity will also make it easier to modify and adapt the program in the future.

The program should be compatible with as many different operating systems and platforms as possible. Any system specific dependencies or code should be avoided. Installation and start of use should also be made as simple as possible.

Performance efficiency can be measured in term of time-behaviour as well as resource utilisation. As this project does not include development or optimization of the alignment algorithm and instead relies on an existing one, the chances for affecting the performance efficiency are limited to the other components of the program.

4.2 The NCBI's BLAST+ executables

The program's core functionality will rely on the NCBI BLAST+ executables detailed in Table 1. These include the main bulk of the program's algorithms, including the various search functions, search database index file creation as well as querying the database for information.

Table 1. *NCBI BLAST+ executables*

Application name	Application type	Application description
blastn	search	searches a nucleotide query against nucleotide subject sequences or a nucleotide database
tblastn	search	searches a protein query against nucleotide subject sequences or a nucleotide database translated from protein at search time
blastp	search	searches a protein query against protein subject sequences or a protein database
blastx	search	translates a nucleotide query and searches it against protein subject sequences or a protein database
makeblastdb	database	builds a BLAST database
blastdbcmd	database	reads a BLAST database and produces reports

NCBI has available both the source code as part of its C++ Toolkit, as well as the BLAST+ executables. The choice to use the executables as part of LBLASTCLI is based on several reasons. Firstly, the executables have been used by a wide userbase for a long enough period, thus providing informal user testing in addition to any formal testing that was part of their development process. Secondly, they provide a clear interface, where the executables can be replaced with newer versions without touching the source code of the LBLASTCLI. Thirdly, they provide the functionality that LBLASTCLI needed, and any attempt to recreate them as part of this thesis project would have probably resulted in lower quality products.

4.3 The use of C++ over Biopython

Biopython [20] is an open-source project focused on creating non-commercial Python tools for computational biology and bioinformatics. There are modules that provide object representation of biological sequences with alphabets, such as nucleotide and protein sequences, support for the most common file types including FASTA, sequences container parsers as well as interfaces for invoking the various NCBI Blast tools.

However, the interfaces such as `Bio.Blast.Applications` only provide wrappers for calling the C++ based BLAST+ executables. The sequence object modules such as the

Bio.SeqIO would very useful for more complex data processing, but they are not necessary for LBLASTCLI. Due to this, in addition to possible gains in performance efficiency, it made sense to use C++ for the whole program.

4.4 The use of concurrency

With any program or system where performance efficiency, and specifically time-behaviour, is of interest, adding concurrency into the architecture, or at least making the program thread safe and possible to run concurrently, is one solution. The program can be made to run concurrently on multiple levels from simply running separate searches (with separate inputs) as concurrent processes, to using concurrent threads in the implementation of the BLAST algorithm or I/O operations.

From the BLAST+ executables the search applications include an optional parameter num_threads to specify the desired number of threads (CPUs) to use in the BLAST search. Automating this to run at the maximum available number supported by the system the program is being executed in would provide maximum efficiency to the user without adding any complexity to the user interface. However, it would seem that the multi-threading is only used for a portion of the BLAST search [21]. Based on some informal tests that others have done [22], multi-threading provides a modest performance boost up to a certain point. In contrast, splitting the query data and blasting it in parallel processes would offer a much more significant reduction in run time.

5. RESULTS AND ANALYSIS

In this chapter we go over the results of the project in the form of the software product, any problems or other unexpected things that came up in the development process, as well as discussion on potential future development ideas.

5.1 Overview of the program architecture

The program was written in C++ with QtCreator 5.9.6. on Red Hat Enterprise Linux Workstation 7.6. LBLASTCLI is a terminal or command line invoked program with only one use case; run a specific BLAST search of the query file input against the target file(s) input and create as a result files containing the sequences from the target input that best match each sequence in the query file. The user can choose from several different BLAST variations and provide a cut-off *E-value*. The optional argument `-limit` allows the user to set the maximum number of sequences that are included in each result file. The option flags `-verbose` and `-timed` provide printed progress messages for the program and a time log of how long each general process (command line parsing, argument validation, BLAST search and result files writing) takes, respectively. All the options are detailed in Table 2.

Table 2. *LBLASTCLI options, arguments and flags*

Switch	Type	Argument	Description
-h, --help	Optional flag	N/A	Help message
-v, --verbose	Optional flag	N/A	Turn on printing
-t, --timed	Optional flag	N/A	Turn on time logging
--q_file [arg]	Required option	File path	FASTA file with query sequences
--dbs_file [arg]	Required option	File path	Text file with a file path to a FASTA file / line to be used as target sequences
--blast_type [arg]	Optional	{ blastn, blastp, tblastn, blastx } Default = blastn	Type of BLAST search to be performed
--evalue	optional	[0,1) Default = 0.001	E-value cut-off for BLAST search
--limit	optional	Integer default = 100	Maximum number of sequences to include in a result file

The command line parsing was done using an external header-only library cxxopts [23], and the argument validation was done separately. `BlastHandler` is the class that is responsible for interfacing with the NCBI BLAST+ executables to run BLAST and the associated tasks involved, such as creating indexing files for the given target input file, if they do not already exist. The NCBI BLAST+ search applications can produce the alignment results in a variety of different formats and level of detail, but crucially none of them contain the actual full sequences, but rather just the segment pairs for each local alignment. To retrieve the full sequences, either the search database must be queried using the identification information provided by the BLAST search output, or if that is not possible, then by parsing through the original FASTA file that the database files were built from. A simplified module diagram of the program is displayed in Figure 4, and a full one is included in Appendix A.

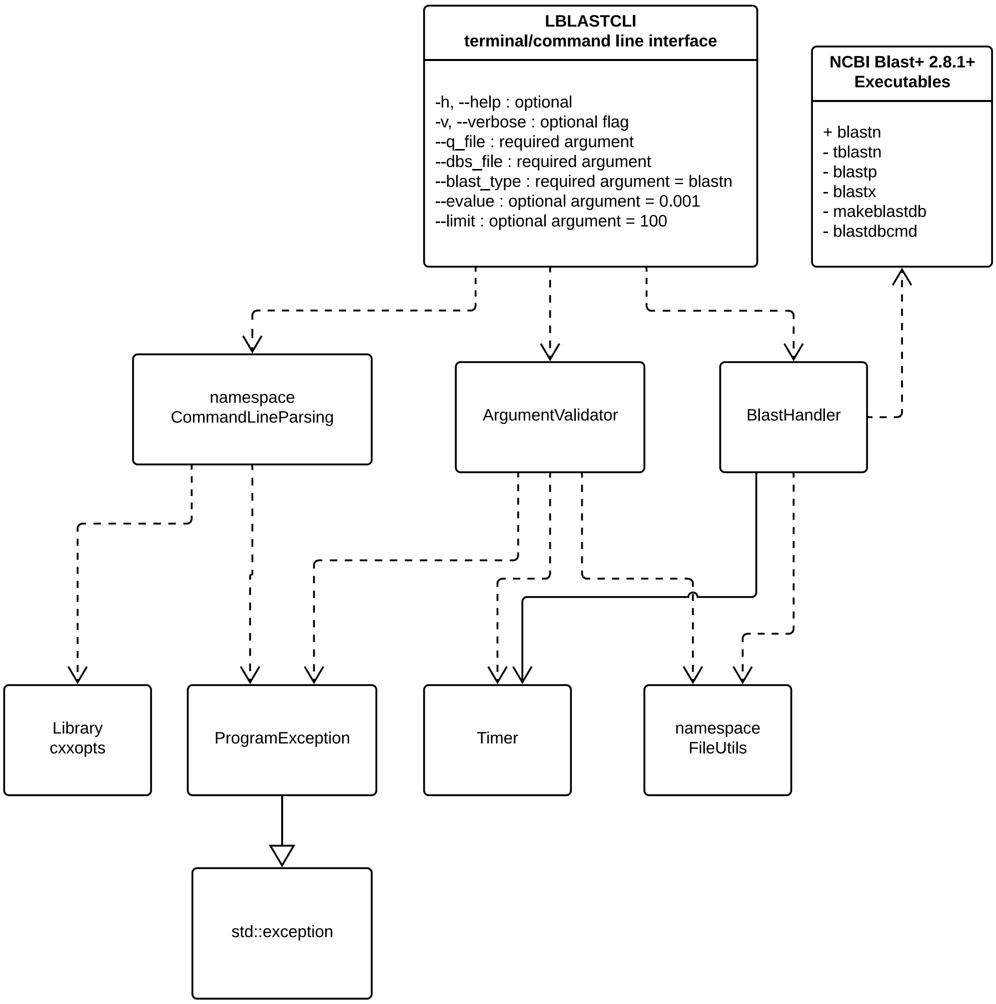


Figure 4. *Simplified class diagram of the LBLASTCLI tool*

The class or module relationships are very simple. The main program uses the `CommandLineParsing` to process the user input from the command line/terminal, then validates the necessary arguments with the `ArgumentValidator` module. After this a

BlastHandler object is created and passed all the arguments from the previous step. Then for each entry in the file path pointed by the argument of the -dbs_file option, BlastHandler::setDBFilePath(std::string fp) and BlastHandler::runBlast() are called. All the tasks associated with the BLAST search, database creation and result file creation are handled internally by the BlastHandler object. The FileUtils module contains functions to check file access and similar tasks, and the ProgramException class is a simple exception class inheriting from std::exception that is used throughout the program. The Timer class provides a simple interface for timing segments of the code and logging their run time along with a message in a log file.

5.2 Quality evaluation

Two of the main quality criteria singled out in the planning of this project, usability and performance efficiency, were not formally tested as part of the project. Planned time-behaviour testing included tests to see the relative performance of using the multi-threading incorporated into the NCBI Blast+ search applications, as well as to compare the performance of full sequence retrieval and writing to the result files from the search database versus the FASTA file. A Timer class was developed and incorporated into the program for this particular purpose. The class has a simple interface that includes methods to start a high-resolution clock, to stop it and get the duration from the previous start, and to automatically log it into a log file along with a descriptive message.

However, the problem turned out to be the formulation of appropriate test data sets. Most publicly available data sets such as the ones provided by NCBI are massive, with even the smaller ones being in the range of 50 000 sequences. Splitting the data set and using a smaller one is an option, but then the problem becomes the choice of matching selection of query sequences to search sequences. How to estimate the number of possible alignment hits based on the input? One could use duplicated sequences in both the query and the target input to ensure a certain amount of hits, but even then, one could not control for the possible alignment across the nonidentical sequences without extremely close inspection of the data. Formulation of appropriate, both from a point of view of being thorough enough as well as reflecting the nature of the real-life cases, test data sets seems to require more domain knowledge than was accounted for in the planning of this project.

In terms of usability, no user testing was conducted. However, the command line interface of the program was very similar to a previously developed Python-based tool, LBlastPyCli [24], for which its users have reported no further problems after the initial stage of learning to use it. In terms of usability, one of the improvements that LBLASTCLI has over LBlastPyCli is that the database creation is incorporated into the program and is done automatically, if the database input files only include the FASTA file and no indexing files. In addition to this, error and exception handling was changed; now users are not directly passed any error messages from the NCBI BLAST+ executables, but rather are shown simplified information more relevant to their use of the program, wth an option to examine the specific information from log files, where the `stdout` and `stderr` streams from the executables are redirected.

The other qualities of modularity, reusability, analysability, modifiability and testability were achieved to a good standard. The overall architecture of the program is very simple with few dependencies between modules, allowing for modification or replacement of them, for example replacing the cxxopts library with another parsing library or module.

5.3 Further development and improvement

There are many areas for improvement and further development. The previously discussed performance efficiency testing and user testing are essential. Basic code level testing in the form of unit testing should also be done. From a usability point of view, any distributed version of the program should include some automatic test scripts to make sure that the program was installed and works correctly.

The program was designed as a simple command line tool so that it could be used directly as well as part of larger analysis pipelines. With the current version the same search type with given parameters such as the *E-value* cut-off is performed on all the query & database pairs given. Allowing the user more flexibility with this would make it easier to use the program to run a variety of searches using varied input types. Surprisingly, the NCBI BLAST+ database application for retrieving information about the database (`blastdbcmd`) does not include an option to check whether the database is a nucleotide or a protein sequence one. This however could be done by simply parsing the file and checking the sequence alphabet (the nucleotide alphabet consists of 5 letters, while the protein one has 20). Automatic matching of the query and search sequence types would eliminate the need for the user to specify the BLAST search type. Another option for doing this would be to incorporate the use of a configuration file; it would be relatively easy to modify the existing program to account for this and allow the user to specify file paths for output files and much more. A more in-depth and specific evaluation of user needs should be done to help further development, so that the program could be improved while keeping it simple and efficient.

6. CONCLUSIONS

This thesis set out to document the use of computational tools built on the BLAST algorithm and to cover the development of a simple and efficient BLAST-based command line tool (LBLASTCLI) that fit a specific use case identified within the user base. The first chapter covered the basic theory behind sequence alignment and the BLAST algorithm as well as existing computational tools. The next part covered software quality and its role in bioinformatics software to give some perspective to the quality criteria selected for our project. After setting the desired quality criteria and technical parameters of the project, the program was developed, and the results discussed. The program that was produced fits the desired functional specifications, but some other planned aspects, that would have supported the desired quality criteria, such as testing, were not achieved. The end of the previous chapter included discussion of some of the technical improvement ideas.

REFERENCES

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, Nov. 1990.

[2] P. Prins, J. de Ligt, A. Tarasov, R. C. Jansen, E. Cuppen, and P. E. Bourne, “Toward effective software solutions for big biology,” *Nat. Biotechnol.*, vol. 33, p. 686, Jul. 2015.

[3] J. D. WATSON and F. H. C. CRICK, “Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid,” *Nature*, vol. 171, no. 4356, pp. 737–738, 1953.

[4] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. 1998.

[5] Z. Zhang and M. Gerstein, “Patterns of nucleotide substitution, insertion and deletion in the human genome inferred from pseudogenes,” *Nucleic Acids Res.*, 2003.

[6] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J. Mol. Biol.*, 1970.

[7] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *J. Mol. Biol.*, 1981.

[8] S. Altschul *et al.*, “The anatomy of successful computational biology software,” *Nat. Biotechnol.*, vol. 31, no. 10, p. 894, 2013.

[9] G. D. Schuler, J. A. Epstein, H. Ohkawa, and J. A. Kans, “[10] Entrez: Molecular biology database and retrieval system,” 1996, pp. 141–162.

[10] “NCBI C++ Toolkit.” [Online]. Available: https://www.ncbi.nlm.nih.gov/toolkit.

[11] “International Organization for Standardization.” [Online]. Available: https://www.iso.org/home.html. [Accessed: 27-Apr-2019].

[12] “ISO/IEC 25010.” [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Accessed: 27-Apr-2019].

[13] “ISO/IEC 25023:2016.” [Online]. Available: https://www.iso.org/standard/35747.html. [Accessed: 27-Apr-2019].

[14] “CISQ Supplement to ISO/IEC 25000.” [Online]. Available: https://it-cisq.org/cisq-supplements-isoiec-25000-series-with-automated-quality-characteristic-measures/. [Accessed: 27-Apr-2019].

[15] M. Umarji, C. Seaman, A. G. Koru, and H. Liu, “Software engineering education for bioinformatics,” in *22nd Conference on Software Engineering Education and Training*, 2009, pp. 216–223.

[16] D. Counsell, “A review of bioinformatics education in the UK,” *Brief. Bioinform.*, vol. 4, no. 1, pp. 7–21, Mar. 2003.

[17] B. Lawlor and P. Walsh, “Engineering bioinformatics: building reliability, performance and productivity into bioinformatics software,” *Bioengineered*, vol. 6, no. 4, pp. 193–203, 2015.

[18] A. H. Kamali, E. Giannoulatou, T. Y. Chen, M. A. Charleston, A. L. McEwan, and J. W. K. Ho, “How to test bioinformatics software?,” *Biophys. Rev.*, vol. 7, no. 3, pp. 343–352, 2015.

[19] T. Seemann, “Ten recommendations for creating usable bioinformatics command line software,” *Gigascience*, vol. 2, no. 1, p. 15, 2013.

[20] P. J. A. Cock *et al.*, “Biopython: freely available Python tools for computational molecular biology and bioinformatics,” *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, Jun. 2009.

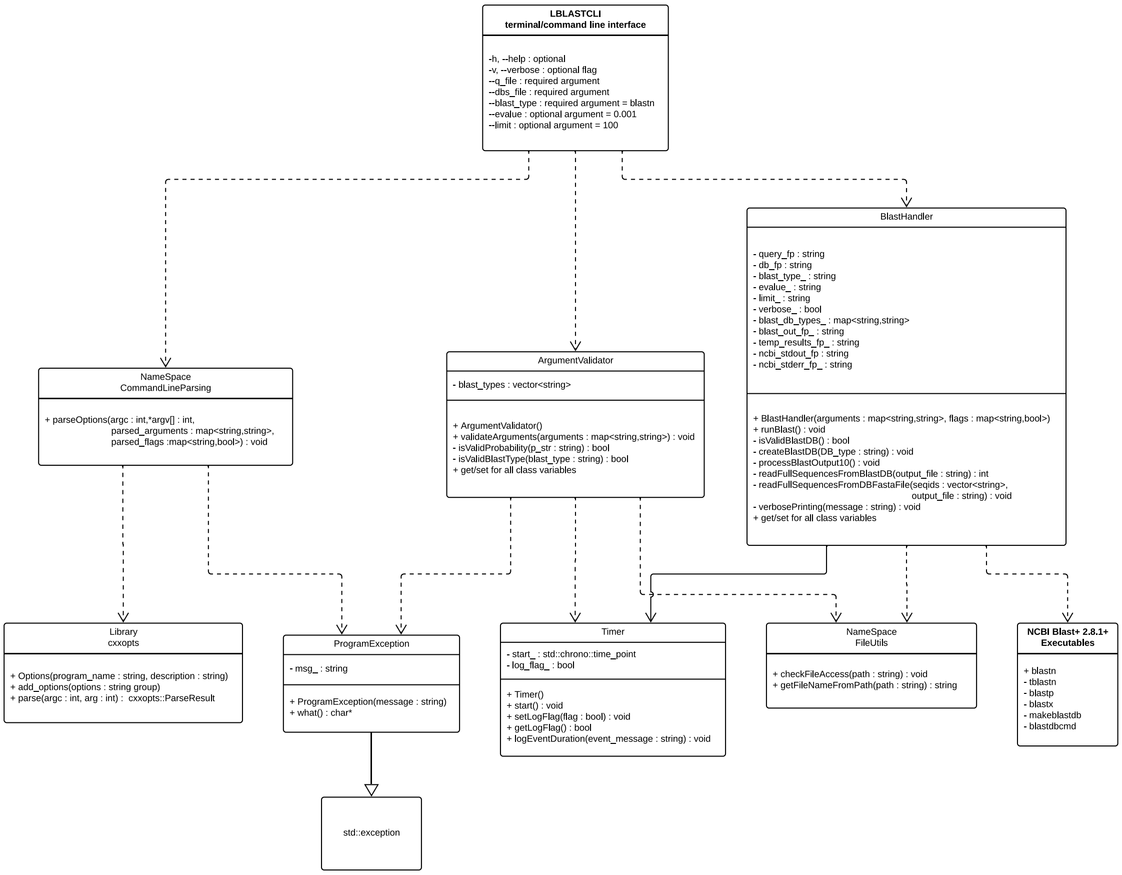
[21] C. Camacho *et al.*, “BLAST+: architecture and applications,” *BMC Bioinformatics*, vol. 10, no. 1, p. 421, 2009.

[22] R. Pascal, “How to correctly speed up Blast using num_threads.” [Online]. Available: http://voorloopnul.com/blog/how-to-correctly-speed-up-blast-using-num_threads/. [Accessed: 28-Apr-2019].

[23] "cxxopts" library. Available: https://github.com/jarro2783/cxxopts [Accessed: 28-Apr-2019].

[24] L. Katariina, “LBlastPyCli” source code. Available: https://gitlab.com/klel/LBlastPycli [Accessed: 28-Apr-2019].

APPENDIX A: LBLASTCLI CLASS UML DIAGRAM


LBLASTCLI
terminal/command line interface
-h, --help : optional
-v, --verbose : optional flag
--q_file : required argument
--dbs_file : required argument
--blast_type : required argument = blastn
--evalue : optional argument = 0.001
--limit : optional argument = 100
BlastHandler
- query_fp : string
- db_fp : string
- blast_type_ : string
- evalue_ : string
- limit_ : string
- verbose_ : bool
- blast_db_types_ : map<string,string>
- blast_out_fp_ : string
- temp_results_fp_ : string
- ncbi_stdout_fp : string
- ncbi_stderr_fp_ : string
+ BlastHandler(arguments : map<string,string>, flags : map<string,bool>)
+ runBlast() : void
- isValidBlastDB() : bool
- createBlastDB(DB_type : string) : void
- processBlastOutput10() : void
- readFullSequencesFromBlastDB(output_file : string) : int
- readFullSequencesFromDBFastaFile(seqids : vector<string>,
output_file : string) : void
- verbosePrinting(message : string) : void
+ get/set for all class variables
ArgumentValidator
- blast_types : vector<string>
+ ArgumentValidator()
+ validateArguments(arguments : map<string,string>) : void
- isValidProbability(p_str : string) : bool
- isValidBlastType(blast_type : string) : bool
+ get/set for all class variables
NameSpace
CommandLineParsing
+ parseOptions(argc : int,*argv[] : int,
parsed_arguments : map<string,string>,
parsed_flags :map<string,bool>) : void
Library
cxxopts
+ Options(program_name : string, description : string)
+ add_options(options : string group)
+ parse(argc : int, arg : int) : cxxopts::ParseResult
ProgramException
- msg_ : string
+ ProgramException(message : string)
+ what() : char*
std::exception
Timer
- start_ : std::chrono::time_point
- log_flag_ : bool
+ Timer()
+ start() : void
+ setLogFlag(flag : bool) : void
+ getLogFlag() : bool
+ logEventDuration(event_message : string) : void
NameSpace
FileUtils
+ checkFileAccess(path : string) : void
+ getFileNameFromPath(path : string) : string
NCBI Blast+ 2.8.1+
Executables
+ blastn
- tblastn
- blastp
- blastx
- makeblastdb
- blastdbcmd