Bioinformatics Algorithms In-course Assessment 2025 -BLAST

Simon Tomlinson March 2025

This assignment is based around the class code for the BA7 class on the topic of **BLAST database searching and the Blast101 application**. But is also draws upon many of the algorithms we have covered in the course so far.

There are three parts to this ICA.

1. Build a command line interface for the Blast101 programme. [40% marks]
2. Testing the Blast101 application – writing tests for the application [50% of the marks]
3. Answer extra questions on BLAST searching [10% of the marks]

Part 1- Building the Command Line Interface

In class we explored how to write a command line interface. This could be used to pass parameters to Python code that runs the BLAST search. The aim here is that if you control the parameters that the programme takes in to run, then you can intercept many possible errors that can happen when the code runs. The most common execution problems come from issues with the input data. So by controlling this in a command line, this effectively can act as a "gatekeeper" for the rest of the code. The command line can intercept incorrect inputs early, before bad things happen later in the running code. Once a command line interface is in place, the code can be used outside the IDE.

The Blast101 code is quite complex, and BLAST takes many parameters which are fine-tuned to make sure the heuristic search gives the required performance- balancing speed against sensitivity of the search. The code contains a settings file (settings.ini) is used to store these settings. These settings are persistent- they are stored on disk between runs of the programme. I suggest that the command line just sets the most important parameters (query sequence, database etc) and keeps the rest of the parameter settings to be picked up from settings.ini. The command line does not need to ask the user to enter all the different parameters- just the absolutely essential ones. This mimics the NCBI BLAST web page- most users do not change the detailed BLAST parameter settings- users just focus on the main information to enter (eg the sequence to be searched and the database to search) and the default settings for the parameters are then used.

What would be useful from the command line would be to check the sequence and database (etc) are valid. So the format of the sequence and the residue composition could be checked. If for example, a user accidentally enters DNA sequence, not protein sequence, then this could be identified before a search is run on the incorrect input.

It is also useful to print the state of the input variables before the code runs.

Another useful feature would be the ability to swap between running the BLAST search to running Smith Waterman search or running the code for the statistical analysis. At the moment there are many entry points into the code, and these are run by selecting one or other of the files in PyCharm, using right click to access the menu for that file and selecting Run. The command line interface manages this decision and therefore allows code to run outside PyCharm.

Please make sure your command-line interface presents a welcoming, clear interface to users. It is up to you how you achieve this.

Part2 – Writing Tests

When you first start coding, the focus is on generating code that is syntactically correct, code that is then recognised as correct Python code by the interpreter and is then able to be run. Writing clearly commented, well organised code is helpful for spotting bugs, and making use of version control approaches can make sure that changes that happen to code over time can be catalogued and kept track of. We will never eliminate the occurrence of bugs in code, but we can follow techniques that allow us to minimise the risk and mitigate common problems.

The Blast101 code that runs some basic tests, but the code is not super-well organised, and more tests are needed! In order to write tests, we need to know the expected output. For example, we need to know the expected score of an alignment. Most often this can be obtained for a simple example, such as a manually calculated alignment score. Or we can use existing tools (eg water) to check our alignments, as long as we can somehow get these tools to mimic our scoring scheme. For water this is tricky for the gap scoring we use, but we might still use a sequence without gaps in the top scoring alignment to test. Any testing is better than none and more testing is always better!

It is important that these tests need to be automatic- we want this all to happen in code without manual intervention. Then ideally, we can specify on the command line if we want to run tests. If we run the tests, then these should be automatic and return if each test was passes or failed. The programme can also return why tests were failed to help localise the issue with the code.

Part3- Extra Questions

Can you consider the following questions and write a short paragraph to answer each

1. When the code counts all the residues in the library, some residues returned are not to the residues counted in the BLOSUM table. What do you think these extra residues represent and how will the BLOSUM62 table deal with these? What will be the impact on the alignment? Should we remove these residues form the random alignment simulation or will this cause bias?
2. Many proteins have internal repeat structures. How do you think these will be treated in the search? Can we tweak the code to improve our ability to find these repeats if they are conserved?
3. If we want to really speed up the search, we could cluster the database and then search through this clustered data. The idea is that close matches (eg all these nanog genes) are close together in overall sequence in the database. So when we find all of these hits by matching to a representative sequence of the group and then just looking at all the similar sequences. What do you think of this strategy? Will it cause any problems with our searche?

Marking Scheme and Submission of the Work

The idea of this assignment is that when we see small snippets of code as an example, it is hard to see how these come together in a complete application. This example code shows how the algorithms we learn in class can be combined together to build a larger application. But it can be tricky to work with lot of code, especially if written by someone else. It is important to note that you do not have to understand every line of the code. An outline understanding is fine, as long as you can construct tests and write the command line interface.

Also note that the code is still very new and has not yet been fully tested. Your code will provide an extra level of testing that is really needed. You might well find some bugs!

The coding tasks are open ended, but you should prioritise the functionality that you think is most important. The 'law of diminishing returns' applies here. For example, in the command line interface there is little value to add commands that change parameters other than the most important ones. These parameters are set in the settings.ini file and most users will not value changing these at the command line. Adding functionality supporting changes these parameters will not attract a lot of marks compared to adding the most important functionality.

You should submit a report of **up to 6 pages** outlining how you have addressed these three topics and also submit your final code as a zip archive. Do not submit a link to the code on GitHub or other such code repository- please submit the actual code.

Use of AI & Citing Sources

You are welcome to use AI to support writing your code. However, do not submit code written by AI as part of the answer to your assignment. Use the AI to provide some ideas and guidance. Tools such as ELM are good at providing 'code snippets' but do not write complete applications. They also represent the consensus view, which is quite often wrong or out of date. This is quite tricky to deal with when writing algorithms. For example, most of the BLAST descriptions online are reasonable as outlines but because BLAST has changed quite a lot in details over the years, these may not represent the current algorithm. LLMs will mix and match components from different generations of the algorithm causing a lot of confusion. Hybrid algorithms often will not achieve the required task. So LLMs work well with simple unchanging but popular algorithms (eg Smith Waterman) but this does not extend to all algorithms. When we code algorithms, we need to precisely understand how the algorithm or we will not be able to code the algorithm. This requires a more detailed understandind than LLMs can typically provide.

If you use any code snippets from other sources, please make sure you link to the original source or provide a full citation. These links can be in the comments around the code. There are some examples of how to do this in the Blast101 code…