Blastx: Implementation of a BlastX variation

Marjan Faizi

Department of Computer Science, Free University of Berlin, Takustr. 9, 14195 Berlin, Germany
E-mail: *marjan@fu-berlin.de*

ABSTRACT

Metagenomics has to handle a large amount of sequence data from environmental samples produced by Next Generation Sequencing. Of particular importance is to find protein-coding genes and their biological functions, where the large amount of data causes problems.
Therefor we developed a BlastX variation to compare the sequence data with a given proteindatabase. Our tool can create different reduced amino acid alphabets and constructs an index for the seeds of flexible length as well as for the proteindatabase to find matches.
This tool should offer faster protein similarity search than the normal BlastX due to the two indices.

1 INTRODUCTION

Next generation sequencing methods have improved considerably the genome sequencing. These new techniques enable to produce a huge amount of short DNA reads within a couple of days, which have to be analysed by domains like metagenomics. Therefor fast tools are much-needed to identifying protein families.

There are already programs like BlastX who does this, so translates reads in all six frames to search them afterwards in the database. We tried to implement a rapid variation of BlastX by constructing simultaneously two indices, one for the seeds and one for the database. To find matches both indices will be put on top of each other and compared, that way protein similarity search is much quicker.

In addition we use reduced amino acid alphabets to translate all reads. This has beneficial effects, since some amino acids have same chemically characteristics, substitute them is non-effective related to their biological functions and we can find so similar proteins even though one of them changed due to mutation in the nucleotid sequence.

2 IMPLEMENTATION

First of all our program translates short DNA reads in six frames by using a reduced alphabet, the proteindatabase will be translated with the same alphabet. After that all reads will be seperated in seeds of flexible length. Then we search (exact or with mismatches) for seeds in the database. Finally we display only the verified matches.

2.1 Translate reads with reduced amino acid alphabets

We implement two different methods to build reduced amino acid alphabtes. The first method GET_ALPHABET_FORCE() groups all amino acids according to their chemical similarity and the bonding forces they develop (e.g. hydrogen bond, ionic bond, etc.), after all the reduced alphabet, created with this method, will have five different characters plus the stop codon at the end.

The second method GET_ALPHABET() clusters amino acids on the basis of scoring matrices. In the SeqAn library are eight matrices stored, we attached them all in our program so the user can decide which one he will take. Furthermore one can choose how many different characters the alphabet should have. GET_ALPHABET() search for the maximum in the given scoring matrix and groups the associated amino acids as long as there are as many characters as the user supplied. At the end the stop codon will be added to the alphabet so that we have always one more character than assigned to the method.
If there are no parameters supplied to the tool the method DEFAULT_VALUES() will set them with values of which we think are fitting the best.

2.2 Search for seeds in database by using indices

After translating the reads they will be seperated in short seeds and we search for them in the database. As already mentioned, we build indices for seeds and proteindatabase, which happens parallel, to compare them and find so matches.

2.3 Verify matches

Only verified matches will be displayed in a .txt-file. So in conclusion we evaluate these matches by calculate their e-value E:

$$E = K * d * m * n * e^{-\lambda * s} \text{ [1]}$$

Where d is the number of sequences in the database, m the read length, n the protein length and s the score of the alignment. The other parameters are constants, at which K = 0,035 and $\lambda = 0,252$ is [2].

Table 1. Recording run-time

Alphabet size	running time [cpu time] Blosum42	Pam200	Vtml200
7	269120000	267710000	267810000
9	264870000	263400000	263440000
11	264260000	259930000	257590000
13	264310000	264620000	264820000
15	291710000	287680000	288400000
17	417700000	415220000	418430000
19	425240000	429130000	429710000

To decide wether an alignment is statistical significant or not, we choose a treshold from 0.001 [3].

3 RESULTS

I searched for 1000 reads in a database with 9406 proteinsequences, to test the run-time of our program with a seed length of 15 bp, no mismatches and with a flexible alphabet size (number of different characters in an alphabet). Also there are no mutated reads in the test case. Furthermore the reduced alphabet created by GET_ALPHABET_FORCE() and three matrices were tested: Blosum45, Pam200 and Vtml200. All values from these scoring matrices are listed in Table 1.

The run-time of each scoring matrix decreases with the increasing alphabet-size until an alphabet size of 13, where the run-time rises rapidly. Most of the time the scoring matrix Pam200 has the lowest value, but a size of 11 the Vtml200 matrix is the fastest one. With each matrix we find exact all matches except with Pam200 and an alphabet size of 7, which finds four more matches. The reduced alphabet created by the method GET_ALPHABET_FORCE() causes a run-time of 270190000 and finds much more matches. Also one can note that the method FIND_MATCHES() takes the longest time to finish.

4 CONCLUSIONS

A seed length of 15 bp proved the best, since lower seed lengths caused too much matches.
The run-time with the alphabet created by GET_ALPHABET_FORCE() was probabily so high because of the small number of different characters.
Based on our results, a protein similarity search perfomed by our tool would be best and without losing any sensitivity, with an alphabet size of 11 bp and the scoring matrix Pam200. Even though the Vtml200 matrix had a better run-time at an alphabet size of 11 bp, Pam200 was predominantly better. By checking the alphabet size of 11 bp more than once, the value of the Vtml200 matrix turned out to be a random outcome.
As we create the reduced alphabet once, we ignored it during benchmarking. In this process the method FIND_MATCHES() took most of the time to complete so its important to turn one's attention to this if the run-time should be improved generally, like we did by building indices for the seeds and the database.
Concerning the testing of our program, it would have been nice to see how many matches the tool finds if some of the reads are mutated.

5 REFERENCES

[1]
http://www.ncbi.nlm.nih.gov/BLAST/tutorial/
[2]
http://www.bioinformatics.org/pipermail/bbb/2006-February/003000.html
[3]
http://abi.inf.uni-tuebingen.de/Teaching/Old/WS2009/BIBC/folien-1/BIBC_WS09_07_Datenbanksuche_3up.pdf