MySQL protein sequence similarity search plugin

With a minor source code modification of MySQL's fulltext search, MySQL can be used to search a protein sequence in large database of protein sequences. Protein sequence similarity search can be incorporated into a running MySQL-sever by installing a plugin. This document contains a brief explanation why natural language fulltext search can be easily converted into sequence similarity search, instructions about installing and testing the plugin, comparison of search results and speed with BLAST and the modified section of the source code including comments.

1 Introduction

1.1 Motivation

Sequence similarity search is arguably the most important tool in bioinformatics. Classical pairwise sequence similarity searches such as blast search a database for the sequences most similar to the query sequence on demand. They can be used via online-service or as downloaded program and are efficient and easy to use on a desktop computer for small numbers of query sequences. However, large scale analyses such as whole genome annotation or finding orthologous groups require considerable computational resources and time. To address this issue, SIMAP provides a pre-calculated protein similarity matrix of the entire protein sequence universe [1] [2]. Instead of calculating large numbers of sequence similarities, protein similarity networks can be downloaded from SIMAP. However, if a given query protein sequence is not present in SIMAP (despite large coverage), the most similar protein sequence present in SIMAP is used instead. For small queries, this can be the rate limiting step. For this step, only the identity of the most similar sequence in SIMAP is needed - neither an alignment, and E-value or additional similar sequences are required. The MySQL fulltext search is already optimized for finding a query text - or the most similar text - in large databases. Modifying this tool to for sequence similarity instead of text similarity could therefore be useful for the SIMAP project.

1.2 Using MySQL for protein sequence similartiy search

MySQL offers fulltext search for natural language. The Text of all fields of the table to be searched as well as the query text are parsed into words. In case of natural language, a word is defined as a white space separated string. Each field of the table is assigned a score corresponding to the match between the text of this field and the query sequence. The more words are shared by the query sequence and the field, the higher the score. Frequent words (words that are found in many other fields as well) contribute less to the overall score. Similarly, alignment tools such as BLAST or FASTA parse a protein sequence into k-mers and search these k-mers in a database. It can be seen that fulltext search and sequence comparison are inherently similar. Therefore, sequence similarity search can be implemented with MySQL with only minor modifications of its built-in fulltext search. Using the built-in MySQL fulltext search for finding a given protein sequence in a database would already work if an identical sequence was present in the database. Both query and target sequence would be considered as one word by the mysql fulltext-parser as they do not contain any white spaces and a match would be returned. However, to find the most similar target sequence in absence of sequence identity, the MySQL-fulltext parser has to be modified to parse the sequence into k-mers. Basically, only MySQL's definition of a word as a whitespace separated string (of arbitrary length) hast to be changed. Instead, the modified parser parses a sequence of length N into (N - k) overlapping k-mers of length k. MySQL then assigns a score for protein sequence in the database according to the number of identical k-mers it shares with the query sequence. More frequent k-mers contribute less to this score. With this minor modification,

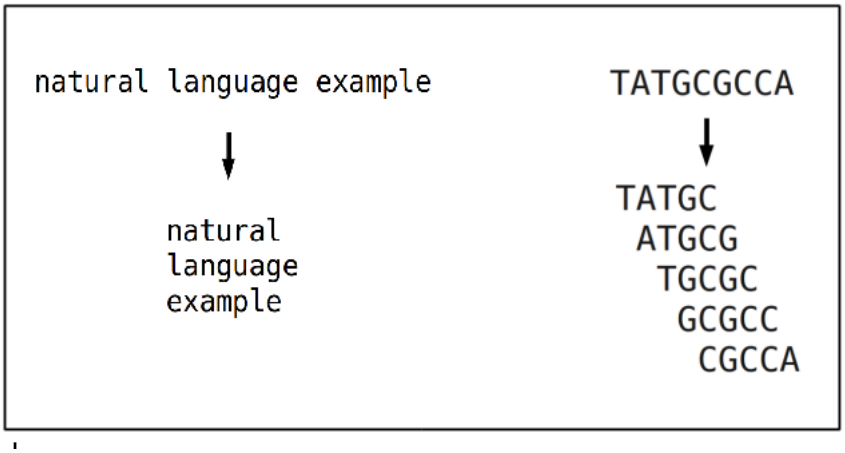

Figure 1: Comparison between built in fulltext parsing and k-mer parsing used by the plugin.

MySQL can identify the sequence that is most similar to a query sequence.

2 Installation guide

The plugin can be installed in three ways: (I) by downloading the pre-compiled plugin and adding it to a running mysql-server, (II) by building MySQL from source including the source code file of the plugin or (III) by compliling the plugin from source separately and adding the compiled plugin to a running server. The first option is the quickest and easiest and is recommended when in doubt while the other two option allow you to make changes in the source code before using the plugin. Note on operating systems: The plugin has been developed and tested on a linux operating system and has been shown to work for both Linux Mint and CentOS. Unfortunately, this installation guide does not contain information on how to compile the plugin in a Windows or Mac operating system.

2.1 Quick installation guide

1) Download the compiled plugin "plugin_example.so" from Github [insert link]
2) Enter the MySQL-command

```
SHOW VARIABLES LIKE 'plugin\_dir;'
```

in MySQL to find out the plugin directory of the MySQL Installation at hand. Then, move the file 'plugin_example.so' to this directory.
3) install the plugin using the MySQL-command:

```
INSTALL PLUGIN kmer\_parser SONAME 'plugin\_example.so;'
```

4) Restart the MySQL-server or reboot and test the plugin (see section test the plugin)

2.2 Include the plugin source code in a MySQL installation from source

1) Download the source code file 'plugin_example.c' from [bithub] and optionally modify it to suit your needs.
2) After you have downloaded your MySQL source code, unpack it and go to the directory 'mysql-version'.
3) Issue the bash-command

```
find . -name 'plugin\_example.c'
```

to find the built-in mysql fulltext parser plugin file. This file will probably be found in the directory mysql-version/plugin/fulltext. Remove plugin_example.c (or move it to another location) and instead place the

downloaded plugin_example.c in the directory.
4) Proceed with the source code installation as described in the MySQL-manual
(http://dev.mysql.com/doc/refman/5.6/en/source-installation.html).
5) Install the plugin using the MySQL-command:

```
INSTALL PLUGIN kmer\_parser SONAME 'plugin\_example.so';
```

6) Restart the MySQL-server or reboot and test the plugin (see section test the plugin)

2.3 Compile the plugin separetly

1) Download the source code file plugin_example.c from [bithub] and modify optionally modify it to suit your needs.
2) Additionally, you need to include header files for the compilation which can be found in the directory include of the MySQL source code. If you already have a MySQL sever built form source, you can search for the path to this directory in the root directory. Alternatively, you can download and unpack the MySQL source code and find the include directory in the directory mysql-version.
3) Go to the directory where the source file (plugin_example.c) is and issue the bash-command

```
gcc -DMYSQL\_DYNAMIC\_PLUGIN -I/path/to/include -shared -fPIC -o plugin\_example.so plugin\_example.
```

There should be no error messages and a SO-file (plugin_example.so) should have been created. (There are certainly other ways to compile the plugin and some may be easier.)
4) Copy the newly created file plugin_example.so to the plugin directory as described in the quick installation guide.

3 Test the plugin

3.1 MySQL-commands for testing

```
1) CREATE DATABASE plugin\_test;
# make database
2) USE plugin\_test;
# use the newly created database
3) CREATE TABLE test ( ID VARCHAR(500), seq VARCHAR(40000) ) ENGINE=MyISAM;
# make table with two columns: one for the sequence ID and one for the sequence
4) LOAD DATA LOCAL INFILE '/path/to/testseq.fa' INTO TABLE test FIELDS TERMINATED BY ''
LINES TERMINATED BY '';
# load fasta-file into table
5) SELECT * FROM test limit 10;
# inspect the fist 10 entries of the table.
6) ALTER TABLE test ADD FULLTEXT INDEX ft\_index(seq) WITH PARSER simple\_parser;
# create fulltext index using the plugin
7) SHOW INDEX FROM test;  \\# to check whether an index has been created
8) SELECT ID, MATCH(seq) AGAINST('') AS RELEVANCE FROM test ORDER BY RELEVANCE DESC LIMIT 10;
# insert query sequence after AGAINST to select the IDs of the 10 most similar sequences
```

3.2 Use a single sequence fasta file to test for the basic functionality of the plugin

1) Search for the first 10 amino acids. The sequence should be found.
2) Search for the first 9 amino acids. Query sequences of less than 10 amino acids should not return a result when the K-mer length is defined as 10.
3) Search for subsequences of varying length. The score should be higher, the longer the query sequences.

3.3 Use the pdb database to compare the search results to blast

1) Create another table, load the pdb database in fasta format into this table and create a fulltext index with the plugin parser as described above.
2) Use any sequence that is not present in the pdb database as query sequence.
3) Use the same sequence in a blast search against the pdb database. This can be done either via the web-service or a locally installed blast program. Note that the blast search has to be done against the same pdb release that has been imported into MySQL. Otherwise blast may find sequences that are not present in the MySQL database thus leading to a false negative.
4) Repeat the process with a few query sequences and check whether the highest scoring result from the MySQL search is among the highest scoring homologues in the blast search.

4 Use the plugin

Load a database of interest into MySQL and calculate a fulltext index with the plugin parser using the MySQL-commands described above. Note that the index calculation can take up to serveral hours and even days depending on the database size. Also note that the index file will me more than ten times the size of the database file. It is recommended that the RAM available be big enough to hold both the database and the index file in memory.
To enable quick searches, the index file has to be in memory. Issue the MySQL-command

```
SHOW VARIABLES LIKE 'datadir';
```

to find the data-directory. Within this directory there will be a directory with the name of the database. Copy this directory into memory with the bash-command

```
cp -r my\_nr /dev/shm
```

Then, rename the original directory or copy it to another location as a backup. Create a symbolic link to the directory by issuing the command

```
ln -s /dev/shm/my\_nr
```

in the MySQL data-directory. Finally make sure that the all files in the directory /dev/shm/my_nr belong to the user mysql by executing the command

```
chown mysql:mysql *
```

within this directory. After this, the plugin will be ready to use.

5 Source code

The section of the plugin source code that has been modified to enable protein sequence similarity search is shown below. For a detailed discussion of the whole plugin source code please refer to the corresponding section of the MySQL-manual (http://dev.mysql.com/doc/refman/5.6/en/writing-full-text-plugins.html). Comments from the MySQL developper are denoted with slash-asterisk ("`/*...*/`") while comments regarding the modification are denoted with double-slash ("`//...`"). k is defined at the beginning of the source code. The function "simple_parser_parse" then parses a protein into overlapping k-mers of length k (see Figure 1). A k-value of 10 has been found to enable quick searches. Each k-mer then passed on to the function "add_word" just like a natural language word in the original fulltext parser plugin. However, in this function the k-mer is translated into a reduced amino acid alphabet. The grouping of functionally related amino acids increases the number of identical k-mers (of a given size) between homologous proteins without loosing much information. Depending on the application, the amino acid alphabet reduction can be omitted (just remove inserted code) or changed to a different alphabet. It is also interesting to note that the plugin could also be applied to nucleic acid sequence search after removing the alphabet reduction and possibly increasing the value of k.

```
#define K 10 // define k-mer length

static void add_word(MYSQL_FTPARSER_PARAM *param, char *word, size_t len)
{
  MYSQL_FTPARSER_BOOLEAN_INFO bool_info=
    { FT_TOKEN_WORD, 0, 0, 0, 0, ' ', 0 };

        // INSERTED CODE BEGIN
        // The piece of code inserted below translates each word (k-mer) into a reduce amino acid
        // alphabet before passing it on to mysql_add_word()
        // amino acid groups of the reduced alphabet used by SIMAP:
        // W            -> W
        // G            -> G
        // H            -> H
        // P            -> P
        // C            -> C
        // FY           -> Y
        // AST          -> A
        // RK           -> R
        // ILVM         -> L
        // DENQBZ       -> D
        // XUOJ*        -> X

        int i;
        char translation_table[256] = {[0 ... 255] = 88}; // The whole array is initialized to 'X'.
        // All characters other in the original k-mer than capital letters will be replaced by 'X'.

        translation_table[65] = 65; // A -> A ('A' in the original k-mer will be replaced by 'A')
        translation_table[66] = 68; // B -> D (...)
        translation_table[67] = 67; // C -> C
        translation_table[68] = 68; // D -> D
        translation_table[69] = 68; // E -> D
        translation_table[70] = 89; // F -> Y
        translation_table[71] = 71; // G -> G
        translation_table[72] = 72; // H -> H
        translation_table[73] = 76; // I -> L
        translation_table[74] = 88; // J -> X
        translation_table[75] = 82; // K -> R
        translation_table[76] = 76; // L -> L
        translation_table[77] = 76; // M -> L
        translation_table[78] = 68; // N -> D
        translation_table[79] = 88; // O -> X
        translation_table[80] = 80; // P -> P
        translation_table[81] = 68; // Q -> D
        translation_table[82] = 82; // R -> R
        translation_table[83] = 65; // S -> A
        translation_table[84] = 65; // T -> A
        translation_table[85] = 88; // U -> X
        translation_table[86] = 76; // V -> L
        translation_table[87] = 87; // W -> W
        translation_table[88] = 88; // X -> X
        translation_table[89] = 89; // Y -> Y

        // Each character (actually the corresponding integer) of a word is used as an index in
        // the array 'translation_table'.
        // The character is replaced by the value of 'translation_table' at this index.
        for (i = 0; i < len; i++) { // charcter of word used as index
                word[i] = translation_table[word[i]]; // replacement
        }
        // INSERTED CODE END

  param->mysql_add_word(param, word, len, &bool_info);
}

/*
  Parse a document or a search query.

  SYNOPSIS
    simple_parser_parse()
      param              parsing context

  DESCRIPTION
```

```
    This is the main plugin function which is called to parse
    a document or a search query. The call mode is set in
    param->mode.  This function simply splits the text into words
    and passes every word to the MySQL full-text indexing engine.
*/

static int simple_parser_parse(MYSQL_FTPARSER_PARAM *param)
{
  char *end, *start, *docend= param->doc + param->length;

  number_of_calls++;

// Original natural language parsing:
// start and end of each word are set to the beginning of doc (text to be parsed).
// Then end is incremented until end is either a whitespace or the end of doc are encountered.
// In either case the word (address of start plus the length of the word (end - start)) is passed to
// the function add_word().

// k-mer parsing (implementet below):
// 'start' and 'end' are placed at the beginning of doc (one sequence).
// 'end' is incremented until either the end of the sequence is encountered or 'end' equals (start+K).
// Then the address of 'start* plus the length of the word (in our case always K) is passed on to
// add_word().
// Finally, 'start' is incremented by one and the process is repeated for the next k-mer.


  for (end= start= param->doc;; end++)
  {
    if (end == docend)
    {
      if (end > start)
        add_word(param, start, end - start);
      break;
    }
    else if ((start + K) == end) // if 'end' is K characters away from start
    {
        add_word(param, start, end - start); // pass another k-mer to add_word()
        start++; // start is moved one residue further to mark the start of the next k-mer
    }
  }
  return(0);
}
```

6 Tests

For testing, a 10% subset of the nr-database (1.6GB) has been used to allow computation of a fulltext-index without the need of a extensive RAM (the server used had 48GB RAM). The k-mer fulltext-index size was 26.7GB which is about 17 times higher than the database file. With a k-value of 10, the k-mer fulltext-index can therefore be expected to be about 17 times the size of the database file. As mentioned before, it is recommended to provide enough RAM to store the entire fulltext index in memory.
Eight protein sequences of lengths 84 to 2550 have been used as query sequences. Seven of those sequences do not occur in the nr-database. Hence, the most similar sequences present in the nr-databases were search by the plugin. The search times (provided the index was copied on the RAM as described in section 4) ranged from 2.07 to 2.43 seconds with an average of 2.18 seconds. It is noteworthy that search times are almost insensitive to query sequence length if the index-file is stored on RAM (with the index file on the hard-drive, the search times are on average 20 times higher and depend strongly on sequence length). The search results have been validated by blastp. For every best matching sequence returned by the MySQL search the blast E-value (using the same query sequence) was rounded to zero by the blast output or very close to zero ($5 * 10^{-27}$ was the biggest E-value).

7 References

References

[1] Arnold, R., Goldenberg, F., Mewes, H. W., and Rattei, T. (2014). SIMAP - The database of all-against-all protein sequence similarities and annotations with new interfaces and increased coverage. *Nucleic Acids Research*, 42:D279–84.

[2] Rattei, T., Arnold, R., Tischler, P., Lindner, D., Stümpflen, V., and Mewes, H. W. (2006). SIMAP: the similarity matrix of proteins. *Nucleic acids research*, 34(Database issue):D252–6.

8 Supplement

8.1 Test query sequences

1) B3KQH5_HUMAN: as present in the nr database
2) B3KQH5_HUMAN: 10 percent of residues randomly mutated
3) BAME1_ARATH: 1 residue mutated
4) BAME1_ARATH: 10 percent of residues randomly mutated
5) MTOR_HUMAN: 1 percent of residues randomly mutated
6) MTOR_HUMAN: 100 residues deleted
7) Q7RSX1_PLAYO: 1 residue mutated
8) Q7RSX1_PLAYO: 10 percent of residues randomly mutated

8.2 Detailed test results

test sequence number	search time [s]
1	2.14
2	2.13
3	2.10
4	2.04
5	2.43
6	2.40
7	2.07
8	2.12
average	2.18