MySQL protein sequence similarity search plugin

With a minor source code modification of MySQL's full-text 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 full-text 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 sequence similarity searches, such as BLAST, search a database for the sequences most similar to the query sequence. They can be used via online-services or as downloaded programs 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 identification of orthologous groups, require considerable computational resources and time. In the example of detection of orthologous groups, the calculation may have to be redone un updated datasets as the number of sequenced genomes increases steadily and rapidly. 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 for a given project ad hoc, 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, finding the most similar sequence among all sequences represented in SIMAP can be time limiting. This step requires only the identity of the most similar sequence in SIMAP - neither alignment, E-value or additional similar sequences are needed. The MySQL full-text 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 similarity search

MySQL offers full-text search for natural language. The Text of all fields of the table to be searched as well as the query text is 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 full-text search and sequence comparison are inherently similar. Therefore, sequence similarity search can be implemented with MySQL with only minor modification of its built-in full-text search. Using the MySQL full-text 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 full-text-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-full-text 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

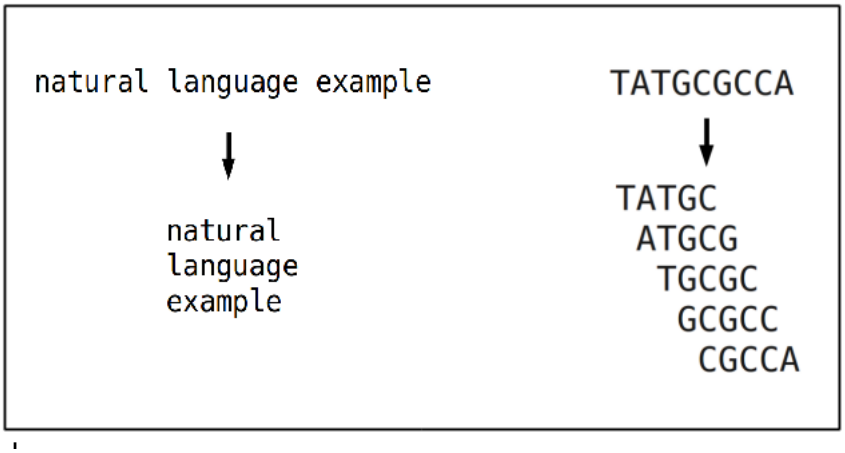

Figure 1: Comparison between built in full-text parsing and k-mer parsing used by the plugin.

changed (Figure 1). Instead, the modified parser parses a sequence of length N into (N - k) overlapping k-mers of length k. MySQL then assigns a score to each 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, MySQL can identify the sequence that is most similar to a query sequence.

1.3 Strengths and Limitations

The plugin has been developed specifically for SIMAP (see section 1.1). However, the plugin or a modified version thereof could also be useful for other applications. This section contains a brief discussion of the advantages and limitations of the plugin to be considered for application.
- Search speed. A pre-calculated index of all k-mers of every sequence in the database enables search times on the order of seconds if the index-file is held in memory.
- Computational requirements: In order to calculate the index, the RAM available to MySQL needs to be large enough to hold the entire index. The size of the index can be expected to be 15 to 20 times higher than the size of the database (if k-mers have the default length of 10 residues).
- Relational database managing. As RDBMS (relational database managing system), MySQL allows complex information to be stored in and quickly retrieved from interconnected tables. This feature can be helpful if plenty of annotation has to be included in a protein sequence database. The plugin adds sequences similarity search to the standard SQL features allowing to retrieve specific annotation for sequences similar to a query sequence.
- Scoring: MySQL's similarity score depends on the number of k-mers shared between the query sequence and a given database sequence as well as the frequency of those k-mers (frequent k-mers contribute less to the similarity score). In contrast to BLAST, the plugin does not create an alignment nor does it offer statistical evaluation of the search results such as BLAST's E-value.

2 Tests

For testing, a 10% subset of the nr-database (1.6 GB) has been used to allow computation of a full-text-index without the need of a extensive RAM (the server used had 48 GB RAM). The k-mer index size was 26.7 GB, which is about 17 times higher than the database file. With the default k-value of 10, the k-mer index can therefore be expected to be about 15 to 20 times the size of the database file. As mentioned before, enough RAM to store the entire full-text index in memory has to be available.
Eight protein sequences of 84 to 2550 amino acids have been used as query sequences. Seven of those sequences do not occur in the nr-database so that the most similar sequences in the nr-databases were found 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 BLASTp E-value (using the same query sequence) was either rounded to zero in the output or very close to zero ($5 * 10^{-27}$ was the biggest E-value).

3 Develpment and source code explanation

MySQL provides an example full-text parser plugin that can replace the built-in full-text parser. The source code for this plugin is very well documented and explained in the manual (http://dev.mysql.com/doc/refman/5.6/en/writing-full-text-plugins.html). Thanks to this excellent documentation it is fairly straightforward to introduce custom changes. Only a small part of the source code had to be modified to enable protein sequence similarity search. The modified section is shown and explained here. Comments from the MySQL developer are denoted with slash-asterisk ("/*...*/") while comments regarding the modification are denoted with double-slash ("//..."). The function "simple_parser_parse" parses a protein sequence into overlapping k-mers of length k (Figure 1). Both, query sequences and database sequences are parsed by this function. K defines the length of the overlapping k-mers ('words') and value of 10 has been found to enable quick searches. Each k-mer is then passed on to the function "add_word" just like a natural language word in the original full-text parser. 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 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 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);
}
```

4 Installation guide

The modified full-text parser plugin can be incorporated in a running MySQL-server. Hence, if you would like to equip any MySQL-server with the protein sequence similarity search, the plugin can be easily installed without the need, to re-install MySQL. The plugin can be installed in three ways: (1) by downloading the pre-compiled plugin and adding it to a running MySQL-server, (2) by building MySQL from source including the source code file of the plugin or (3) by compiling 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.
The plugin has been developed and tested on a Linux operating system and has been shown to work for Linux Mint and Linux CentOS. Unfortunately, this installation guide does not contain information on how to compile the plugin on a Windows or Mac operating system. MySQL version 5.6 has been used for development and testing. Whether the plugin will be compatible with future MySQL versions can only be found out by testing. For information about SQL syntax and MySQL installation that goes beyond the scope of this documentation, it is recommended to refer to the MySQL reference manual (http://dev.mysql.com/doc/refman/5.6/en/index.html).

4.1 Quick installation guide

The easiest way to install the plugin is via the SO-file. This file contains the pre-compiled plugin. All you have to do is to copy this file to the correct directory and execute a command in MySQL. This approach has been tested on MySQL-servers built from source as well as binary installed MySQL-servers so it should not matter how your MySQL-server was installed. However, if you wish to add your own modifications to the plugin, you have to chose one of the other two ways of installation.
1) Download the compiled plugin "plugin_example.so" from GitHub [insert link].
2) MySQL has a special directory for plugins. The path to this directory may between versions. To find the plugin-directory of your MySQL-server, enter the command

```
SHOW VARIABLES LIKE 'plugin_dir';
```

in MySQL.
3) Move the downloaded file 'plugin_example.so' to the plugin-directory.
4) Install the plugin using the MySQL-command:

```
INSTALL PLUGIN simple_parser SONAME 'plugin_example.so';
```

5) Use the MySQL-command

```
SHOW PLUGINS;
```

to show all installed plugins. If the installation was successful, there should be an active plugin named 'simple_parser'.
6) Restart the MySQL-server or reboot and test the plugin (see below).

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

In this approach you build a MySQL-server from source. By replacing the original source file of the full-text parser plugin with the modified one, the new MySQL server will allow sequence similarity search. This approach is recommended if you wish to use MySQL for protein sequence similarity search but do not have a MySQL-server at the moment. Moreover, the downloaded plugin source code can be further modified before the installation if you wish to add your own functionality.
1) Download the source code file 'plugin_example.c' from GitHub [insert link] and optionally modify it to suit your needs.
2) After having 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 full-text 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) After the installation of the MySQL-server, install the plugin using the MySQL-command:

```
INSTALL PLUGIN simple_parser SONAME 'plugin_example.so';
```

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

4.3 Compile the plugin yourself

This way of installation is recommended if you wand to modify the plugin and want to add it to an existing MySQL-server. For the compilation command provided here, GCC (GNU Compiler Collection) has to be installed on your Linux operating system. In addition to the plugin-source code file, header files are required. These can be obtained by downloading a MySQL source code version. This does not mean, that MySQL has to be installed from source; the header files only have to saved somewhere on your hard drive. Make sure to download a source code version of MySQL roughly equivalent to the version of the MySQL-server you would like to equip with the plugin (although it may also work otherwise).
1) Download the source code file plugin_example.c from GitHub [insert link] and 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. You can download and unpack the MySQL source code and find the directory include in the directory mysql-version.
3) Go to the directory where the source file (plugin_example.c) is located and issue the bash-command

```
gcc -DMYSQL_DYNAMIC_PLUGIN -I/path/to/include -shared -fPIC -o plugin_example.so plugin_example.c
```

There should be no error messages and an SO-file (plugin_example.so) should have been created.
4) Copy the newly created file plugin_example.so to the plugin directory as described in the quick installation guide.

5 Test the plugin

After the plugin has been installed, it is recommended to test the plugin on a small database. This ensures that the protein sequence similarity search works as expected for the system at hand and provides an estimate of full-text (k-mer) index size and the time required to calculate it. Moreover, people unfamiliar with MySQL can use this section to obtain an overview of the MySQL-syntax required for using full-text search.

5.1 MySQL-commands for testing

Below, a few MySQL-commands required for using the plugin are provided. These commands explain how to load a protein sequence database in FASTA format into a MySQL table, calculate an index (called 'full-text index' by MySQL but actually a k-mer index in this case), and perform a sequence similarity search. For users familiar with SQL-syntax, this section is not of interest.

```
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 protein-ID and one for the sequence
4) LOAD DATA LOCAL INFILE '/path/to/testseq.fa' INTO TABLE test FIELDS TERMINATED BY '\n'
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 full-text index using the plugin
7) SHOW INDEX FROM test;  # show the newly calculated index
8) SELECT ID, MATCH(seq) AGAINST('') AS score FROM test ORDER BY score DESC LIMIT 10;
# insert query sequence after AGAINST to select the IDs of the 10 most similar sequences
```

Please note that in order to enable full-text search, the engine MyISAM has to be used for the table. The engine can be specified when creating the table (see below) or set as default in the file my.cnf (see http://dev.mysql.com/doc/refman/5.1/en/option-files.html).
The view commands provided here suffice for testing and using the plugin with different databases and the next sections will refer to this set of commands.

5.2 Use a 10-sequences FASTA-file to test the basic functionality of the plugin

The easiest way of testing the plugin is by using a FASTA-file of about ten sequences as a "mini-database". By searching for sub-sequences or mutated versions of one sequence, the correctness of the output can be checked by the user without additional analysis.
1) Create a FASTA-file containing only a few sequences for instance by copying the first ten entries of the Swiss-Prot (UniProtKB/Swiss-Prot) database in a new file. Alternatively, a few protein sequences can be downloaded manually from UniProt and copy-pasted into a single file using a standard text editor. Any protein sequences can be used for this test.
2) Load this mini-database into a MySQL table and create a full-test index as described above.
3) Search for 10 amino acids of one sequence. The sequence should be found.
4) Search for 9 amino acids of one sequence. Query sequences of less than 10 amino acids should not return a result because K is defined as 10 by default.
5) Search for sub-sequences of varying length. The score should be higher, the longer the query sequences.

5.3 Compare the search results to BLASTp

It is recommended to compare the hits found in MySQL with BLAST or another alignment tool confirm the validity of the search results. Any database can be used for the test as long as the same release is used in MySQL and the other alignment tool of choice. However, a small database has the advantage of allowing quick testing. In the example below, the Swiss-Prot database is used for testing.
1) Download the Swiss-Prot (UniProtKB/Swiss-Prot) database in FASTA format.
2) Create a table in MySQL, load the database into this table and calculate a full-text index with the plugin parser as described above.
3) Use any sequence that is not present in the Swiss-Prot database as query sequence. (If a sequence identical to the query sequence is present in the database, this sequences will be found. However, also the built-in MySQL full-text parser could find identical sequences. To test the capability of the plugin to find the most similar sequence(s) in absence of sequence identity, a query sequence not present in the database should be used.)
4) Use the same sequence in a BLASTp search against the Swiss-Prot database. This can be done either via the web-service or locally installed BLASTp. Note that the BLASTp search has to be done against the same database release that has been imported into MySQL. Otherwise blast may find sequences that are not present in the MySQL database (or vice versa), thus compromising the comparison.
5) Repeat the process with a few query sequences and check whether the highest scoring result from the MySQL search is among the closest homologues identified by the BLASTp search.

6 Use the plugin

Load a database of interest into MySQL and calculate a full-text index with the plugin parser using the MySQL-commands. Note that the index calculation can take up to several hours and even days depending on the database size. The index file will me 10 t0 20 times the size of the database file. The RAM available has

to be big enough to hold both the database and the index file.
After the index calculation is completed, the index file has to be copied to the RAM to enable quick searches. 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. For simplicity, copy the whole directory into memory with the bash-command

```
cp -r my_database /dev/shm
```

Note that the RAM may have another path than /dev/shm in your Linux version. 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_database
```

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

```
chown mysql:mysql *
```

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

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] Arnold, R., Rattei, T., Tischler, P., Truong, M. D., Stümpflen, V., and Mewes, W. (2005). SIMAP - The similarity matrix of proteins. *Bioinformatics*, 21: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