proteins, secondary structure
protein similarity, query language

Dominika WIECZOREK[1], Bożena MAŁYSIAK-MROZEK[1], Stanisław KOZIELSKI[1], Dariusz MROZEK[1]

A DECLARATIVE QUERY LANGUAGE FOR PROTEIN SECONDARY STRUCTURES

Searching proteins on their secondary structures provides a rough and fast method of identification of molecules having a similar fold. Since existing database management systems do not offer integrated exploration methods for querying protein structures, the structural similarity searching is usually performed by external tools. This often lengthens the processing time and requires additional processing steps, like adaptation of input and output data formats. In the paper, we present the extended SQL language, which allows searching a database in order to find proteins having secondary structures similar to the structural pattern specified by a user. Presented query language is integrated with the relational database management system and it simplifies the manipulation of biological data.

1. INTRODUCTION

Secondary structures are valuable source of information regarding the construction of protein molecules. This organization level of protein structure allows studying the general shape of proteins and the formation of amino acid chain caused by local hydrogen interactions [1-3]. Representing protein spatial structures by secondary structure elements gives the possibility to reveal and discover types of characteristic spatial elements that are present in the protein conformation [4], [5] – whether there are only α-helices or only β-strands in the structure, or maybe the structure is generically differentiated, what is the arrangement of these elements – whether they are heavily segregated or appear alternately.

Secondary structure representation of proteins became very important in the analysis of protein constructions and functions. It is frequently used in the protein structure similarity searching, e.g. in [6-9]. Observation of three-dimensional protein structure, represented at the level of secondary structures, reveals the mutual spatial organization of individual fragments of the protein and provides information on the formation of various structural motifs (also known as secondary superstructures) [5]. Moreover, observation of protein structures represented by secondary structure elements allows to specify the location of functional domains, which are structurally stable protein fragments that can be folded independently and usually play a particular role in cellular processes. In Fig. 1 we can observe known secondary structure elements: two antiparallel β-strands connected by a short loop in the popular β-hair-pin motif (Fig. 1a) and spiral α-helices in the structure of a sample protein from the Protein Data Bank [10].

For scientists that study the structure and function of proteins, it is very important to have the ability to search for structures similar to the construction of a given structure. This is usually done by external applications, which is a big disadvantage. The main reason of the fact is that data describing protein structures are managed by database management systems (DBMSs), which work excellent in commercial uses. However, they are not dedicated for storing and processing biological data. They do not provide the native support for processing biological data with the use of the SQL language, which is a fundamental, declarative way of data manipulation in most database systems [11]. There are just a few solutions that allow advanced processing of biological data on the database side. However, they are merely prototypes of extensions incorporated into existing DBMSs.

In the paper, we show our extension to the SQL language that allows querying protein spatial structures represented by secondary structure elements. The PSS-SQL language (*Protein Secondary Structure – Structured Query Languaage*) that we have designed and developed supports searching the database against proteins having their secondary structures similar to the structure specified in a user's

[1] Institute of Informatics, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland.

query. The given structure is represented by the structural pattern. In the PSS-SQL, we offer declarative method of protein similarity searching, which is integrated with the database server.

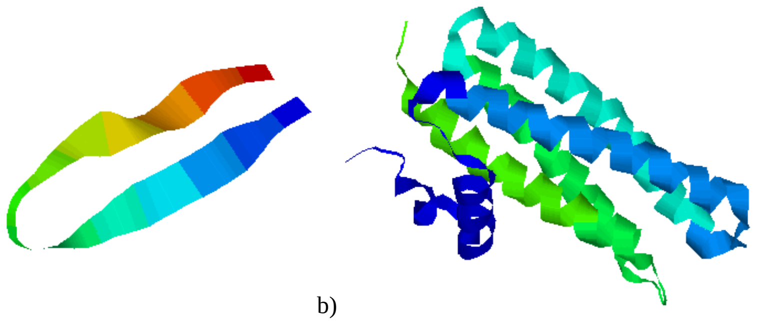

Fig. 1. Sample protein spatial structures represented by secondary structure elements: a) two antiparallel β-strands in popular β-hairpin motif, b) α-helices in the structure of the protein PDB ID: 1X91 (crystal structure of mutant form A of a pectin methylesterase inhibitor from *Arabidopsis*).

2. RELATED WORKS

Having a possibility to store protein structural data in appropriate manner and just submit simple queries to a database would be a great asset for many researchers working in the area of protein bioinformatics. There are only a few initiatives in the world that develop this kind of solutions. They work on different levels of protein structure.

The ODM BLAST [12] is an example of the successful implementation of the BLAST methods in the commercial system and SQL language. The implementation covers declarative similarity searching for DNA/RNA sequences and protein primary structures. It works fast, but it does not cover secondary structure level. The weakness of the implementation is also the necessity to define a cursor inside the query, which complicates the whole construction of the query.

In [13], authors describe their extension to the SQL language, which allows searching on the secondary structures of protein sequences. The extension was developed in Periscope (dedicated engine) and in Oracle (commercial database system). In the solution, secondary structures are represented by segments of different types of secondary structure elements, e.g. hhhllleee. The disadvantage of the solution is the multipart form of the search criteria, which requires hard coding and is not very clear for potential users.

In [14], authors show the Periscope/SQ extension of the Periscope system. Periscope/SQ is a declarative tool for querying primary and secondary structures. To this purpose authors introduced new language PiQL, new data types and algebraic operators according to the defined query algebra PiOA. The PiQL language has many possibilities. However, it can be difficult to use, if someone wants to construct complex search criteria containing many consecutive segments of the secondary structures, especially, with unidentified length or type.

In the paper [15], authors present their extensions to the object-oriented database (OODB) by adding the Protein-QL query language and the Protein-OODB middle layer for requests submitted to the OODB. Protein-QL allows to formulate simple queries that operate on the primary, secondary and tertiary level.

3. DATA FORMAT

Searching similarities in protein structures by formulating queries in PSS-SQL requires that data describing secondary structures of proteins should be stored in a database in a specific format. In the presented solution, we assume that protein structures will be represented by sequences of secondary structure elements (SSE). Each SSE corresponds to one amino acid in the primary structure. In Fig. 2 we show the amino acid sequence of the *Zinc transport system ATP-binding protein adcC* in the *Streptococcus pneumoniae* and the corresponding sequence of SSEs. Particular elements have the following meaning: H denotes α-helix, E denotes β-strand, C (or L) stands for loop, turn or coil.

```
P0A2U6
ADCC_STRPN
Zinc transport system ATP-binding protein adcC OS=Streptococcus pneumoniae GN=adcC PE=3 SV=1

MRYITVEDLSFYYDKEPVLEHINYCVDSGEFVTLTGENGAAKTTLIKASLGILQPRIGKVAISKTNTQGKKLRIAYLPQQIASFNAGFPSTV
YEFVKSGRYPRKGWFRRLNAHDEEHIKASLDSVGMWEHRDKRLGSLSGGQKQRAVIARMFASDPDVFILDEPTTGMDAGSKNEFYELMHHSA
HHHGKAVLMITHDPEEVKDYADRNIHLVRNQDSPWRCFNVHENGQEVGHA

CCCEEECCCEEECCCCCCEEEEEEECCCCCEEEECCCCCCCHHHHHEEEEECCCCCCCCEEEEECCCCCCEEEEEEHHHHHHHHCCCCCCE
EEEECCCCCCCHHHHHCCCCCHHHHHHHHHHHHCCCCCCCCCCCCCCCHHHHHHHHHHHHCCCCEEEECCCCCCCCCCCCCCHHHHHHHHCC
CCCCEEEEEEEECCCCCCCCCCCCCCEEEEECCCCCCCEEEECCCCCCCCCC
```

Fig. 2. Sample amino acid sequence of the protein *Zinc transport system ATP-binding protein adcC* in the *Streptococcus pneumoniae* with the corresponding sequence of secondary structure elements.

Such a representation of protein structure is very simple in terms of storing the structure in a database. Data describing types and location of SSEs in the protein structure may come from different sources – they can be extracted directly from the Protein Data Bank [10], taken from systems that classify protein structures, like SCOP [16] or CATH [17], or generated using programs that predict secondary structures on the basis of primary structures. Nevertheless, they should be represented in the common format as a sequence of H, E, C/L symbols. In our research, we store sequences of SSEs in the *ProteinTbl* table of the *Proteins* database. The schema of the table is presented in Fig. 3.

```
id   protID       protAC name             length primary          secondary
---- ------------ ------ ---------------- ------ ---------------- ----------------
799  ABCX_GUITH   O78474 Probable ATP-... 253    MKKKILEVTNLHA... CCCCEEECCCHHH...
800  1A02_GORGO   P30376 Class I histo... 365    MAVMAPRTLLLLL... CCCCHHHHHHHHH...
808  1A110_ARATH  Q9LQ10 Probable amin... 557    MTRTEPNRSRSSN... CCCCCCCCCCCCC...
809  1A111_ARATH  Q9S9U6 1-aminocyclop... 460    MLSSKVVGDSHGQ... CCCEEEEECCCCC...
810  ABCX_PORPU   P51241 Probable ATP-... 251    MSDYILEIKDLHA... CCCHHHHHHHHHH...
```

Fig. 3. Schema of the table storing protein sequences of SSEs.

Fields of the *ProteinTbl* table have the following meaning: *id* – internal identifier of protein in a database, *protAC* – protein *Accession Number*, *protId* – protein *IDentification* in the SwissProt database, *name* – protein name and description, *length* – protein length in amino acids, *primary* – primary structure of a protein (amino acid sequence), *secondary* – sequence of secondary structure elements of a protein.

4. SSE SEGMENTS INDEXING

Before we start searching proteins that are similar to the specified pattern, we have to build an index on the field storing sequences of SSEs. In this phase, we create additional segment table, which is stored in the structure of B-Tree clustered index. The segment table (Fig. 4) contains extracted information regarding consecutive segments of particular types of SSEs (*type*), their lengths (*length*) and positions (*startPos*). The information accelerates the process of similarity searching through the preliminary filtering of protein structures that are not similar to the query pattern. In the filtering, we extract the most characteristic features of the query pattern and, on the basis of the information in the index, we eliminate proteins that do not meet the similarity criteria. In the next phase, proteins that pass the preselection are aligned to the query pattern.

```
id    protID type startPos length
----- ------ ---- -------- ------
67    3      C    0        3
68    3      H    3        23
69    3      C    26       8
70    3      H    34       12
71    3      C    46       3
72    3      E    49       3
```

Fig. 4. Part of the segment table.

5. PSS-SQL FOR SEARCHING PROTEINS ON SECONDARY STRUCTURES

Protein Secondary Structure – Structured Query Language (PSS-SQL) extends the standard syntax of the SQL language providing additional functions that allow to search protein similarities on secondary structures. We disclose two important functions to this purpose: *containSequence* and *sequencePosition*, which will be presented in the section. However, PSS-SQL covers also a series of supplementary procedures and functions, which are used implicitly, e.g. for extracting segments of particular types of SSEs, building additional segment tables, indexing SSEs sequences, processing these sequences, aligning the target structures from a database to the pattern, validating patterns, and many other operations. The PSS-SQL extension was developed in the C# programming language. All procedures were gathered in the form of the *ProteinLibrary* DLL file and registered for the Microsoft SQL Server 2005/2008.

5.1. PATTERN REPRESENTATION IN PSS-SQL QUERIES

While searching protein similarities on secondary structures, we need to pass the query structure (query pattern) as a parameter of the search process. Similarly to the storage format, in PSS-SQL queries the pattern is represented as a sequence of SSEs. However, the form of the sequence is slightly different. During the development of the PSS-SQL functionality we assumed the new extensions should allow users to formulate a large number of various query types with different degrees of complexity. Moreover, the form of these extensions should be as simple as possible and should not cause any syntax difficulties. Therefore, we have defined the corresponding grammar in order to help constructing the query pattern.

In PSS-SQL queries, the sequence of SSEs is represented by blocks of segments. Each segment is determined by its type and length. The segment length can be represented precisely or as an interval. It is possible to define segments, for which the type is not important or undefined (wildcard symbol ‘?’), and for which the end value of the interval is not defined (wildcard symbol ‘*’). The grammar for defining patterns written in the Chomsky notation has the following form. The grammar is formally defined as the ordered quad-tuple <*N*, Σ, *P*, *S* >:

$$G_{pss} = <N_{pss}, \Sigma_{pss}, P_{pss}, S_{pss}>,$$

where the symbols respectively mean: N_{pss} – a finite set of nonterminal symbols, Σ_{pss} – a finite set of terminal symbols, P_{pss} – a finite set *P* of production rules, S_{pss} – a distinguished symbol $S \in N_{pss}$ that is the start symbol.

```
Σpss    = {c, h, e, ?, *, N+}
Npss    = { <sequence>, <blocks_of_segments>, <segment>, <type>, <begin>, <end>,
            <lenght>, <whole_number_greater_than_zero_and_zero>, <undetermined>}
Ppss    = {
        <sequence> ::= <blocks_of_segments>
        <blocks_of_segments> ::= <segment> | <segment>, <blocks_of_segments>
        <segment> ::= <type> (<begin>; <end>) | <type> (<lenght>)
        <begin> ::= <whole_number_greater_than_zero_or_zero>
        <end> ::= <whole_number_greater_than_zero_or_zero> | <undetermined>
        <lenght> ::= <whole_number_greater_than_zero_or_zero>
        <type> ::= c | h | e | ?
        <whole_number_greater_than_zero_or_zero> ::= N+ | 0
        <undetremined> ::= *}
Spss    = <sequence>
```

Assumption: `<begin> <= <end>`

The following terms are compliant with the defined grammar G_{pss}:

– `h(1;10)` – representing α-helix of the length 1 to 10 elements
– `e(2;5),h(10;*),c(1;20)` – representing β-strand of the length 2 to 5 elements, followed by α-helix of the length at least 10 elements, and loop of the length 1 to 20 elements
– `e(10;15),?(5;20),h(35)` – representing β-strand of the length 10 to 15 elements, followed by any element of the length 5 to 20, and α-helix of the exact length 35 elements

With such a representation of the query pattern, we can start the search process using the *containSequence* and *sequencePosition* functions.

5.2. VERIFYING STRUCTURES USING CONTAINSEQUENCE

The *containSequence* function allows to check if a particular protein or set of proteins from a database contain the structural pattern specified as a sequence of SSEs. This function returns Boolean value 1, if the protein from a database contains specified pattern, or 0, if the protein does not include the particular pattern.

The header of the *containSequence* function is as follows:

```
FUNCTION containSequence
(
  @proteinId int,
  @columnSSeq text,
  @pattern varchar(4000)
) RETURNS bit
```

The *containSequence* function takes the following arguments:

– *@proteinId* – unique identifier of protein in the table that contains sequences of SSEs (e.g. the *id* field in case of the *ProteinTbl*),
– *@columnSSeq* – database field containing sequences of SSEs of proteins (e.g. *secondary*),
– *@pattern* – pattern that defines the query SSEs sequence represented by a set of segments, e.g. *h(2;10), c(1;5),?(2;*).*

The *containSequence* function can be used both in SELECT and WHERE phrases of the SQL SELECT statement. Using the function in the SELECT statement allows to display information, whether the protein or set of proteins contain a specified pattern. Below, we present an example of using the *containSequence* function in order to verify, whether the structure of the *Q9FHY1* protein has the structural region containing β-strand of the length 7 to 20 elements, surrounded by two loops, one of the length 10 to 20 elements, and second of the length of 1 to 20 elements – pattern *c(10;20),e(7;20), c(1;20).*

```
SELECT id, protID, protAC, name,
   containSequence(id,'secondary','c(10;20),e(7;20),c(1;20)')
   AS containSeq
FROM ProteinTbl WHERE protAC='Q9FHY1'
```

Results of the verification are shown in Fig. 5.

```
id   protID       protAC   name                                 containSeq
---- ------------ -------- ------------------------------------ ----------
964  ABIL4_ARATH  Q9FHY1   Protein ABIL4 OS=Arabidopsis thaliana 0
```

Fig. 5. Result of the verification for the protein *Q9FHY1*.

The following query shows an example of using the *containSequence* function in order to display, whether proteins from the *Arabidopsis thaliana* species contain the given pattern (*containSeq=1*) or not (*containSeq=0*). Structural pattern is the same as in previous example.

```
SELECT id, protID, protAC, name,
   containSequence(id,'secondary','c(10;20),e(7;20),c(1;20)')
   AS containSeq
FROM ProteinTbl
WHERE name like '%Arabidopsis thaliana%'
```

Results of the search process are shown in Fig. 6.

```
id   protID       protAC   name                                 containSeq
---- ------------ -------- ------------------------------------ ----------
175  A494_ARATH   P43295   Probable cysteine proteinase A494 OS= 1
244  A9_ARATH     Q00762   Tapetum-specific protein A9 OS=Arabid 0
443  AAH_ARATH    O49434   Allantoate deiminase, chloroplastic O 1
522  AASS_ARATH   Q9SMZ4   Alpha-aminoadipic semialdehyde syntha 1
553  AAT1_ARATH   P46643   Aspartate aminotransferase, mitochond 1
560  AAT2_ARATH   P46645   Aspartate aminotransferase, cytoplasm 1
...
```

Fig. 6. Partial result of the search process for proteins from the *Arabidopsis thaliana* species.

Using the *containSequence* function in the WHERE clause allows to find proteins that contain the specified pattern. Below is an example of using the function for searching proteins from the *Escherichia coli* that contain the pattern *h(5;15),c(3),?(6),c(1;*)*.

```
SELECT id, protID, protAC, name, primary, secondary
FROM ProteinTbl
WHERE containSequence(id, 'secondary','h(5;15),c(3),?(6),c(1;*)')=1
      and name like '%Escherichia coli%'
```

Results of the searching process are shown in Fig. 7.

```
id   protID      protAC   name                   primary                secondary
---- ----------- -------- ---------------------- ---------------------- --------------------------
1294 ACCA_ECO24  A7ZHS5   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHCCH...
1295 ACCA_ECO57  P0ABD6   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHCCH...
1296 ACCA_ECOHS  A7ZWD1   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHCCH...
1297 ACCA_ECOK1  A1A7M9   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHCCH...
1298 ACCA_ECOL5  Q0TLE8   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHCCH...
1299 ACCA_ECOL6  Q8FL03   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHCCH...
1300 ACCA_ECOLI  P0ABD5   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHHHH...
1301 ACCA_ECOUT  Q1RG04   Acetyl-coenzyme A ca... MSLNFLDFEQPIAELEAKID... CCCCCCCCHHHHHHHHHHHHHHH...
```

Fig. 7. Partial result of the searching process for proteins from the *Escherichia coli* having the given structural pattern *h(5;15),c(3),?(6),c(1;*)*.

5.3. LOCATING PATTERNS USING SEQUENCEPOSITION

The *sequencePosition* function allows to locate the specified pattern in the structure of a protein or group of proteins in a database. Pattern searching is performed with the use of segment table and through alignment of protein secondary structures. For this purpose, we have adapted the Smith-Waterman alignment method [18].

The header of the *sequencePosition* function is as follows:

```
FUNCTION sequencePosition
    (  @columnSSeq text,
       @pattern varchar(4000),
       @predicate varchar(4000)
    )
    RETURNS @resultTable table
    (  proteinId int,
       startPos int,
       endPos int,
       length int,
       gapsCount int,
       sequence text
    )
```

The *sequencePosition* takes the following arguments:

– *@columnSSeq* – database field that contains sequences of SSEs, e.g. *secondary*,
– *@pattern* – pattern that defines the query SSEs sequence represented by a set of segments, e.g.: *h(2;10), c(1;5),?(2;*)*,
– *@predicate* – an optional, simple or complex criteria that allow to limit the list of proteins that will be processed during the search, e.g.: *name LIKE '%phosphogluconolactonase%'*,

The *sequnecePosition* function returns a table containing information about the location of the query pattern in the structure of the database protein:

– *proteinId* – unique identifier of protein that contains specified pattern; using the identifier we can join resultant table with data from other tables,
– *startPos* – position, where the pattern starts in the target protein from a database,
– *endPos* – position, where the pattern ends in the target protein from a database,
– *length* – length of the segment that matches to the given pattern,
– *sequence* – sequence of SSEs, which matches to the pattern defined in the query.

The *sequencePosition* function is used in the FROM clause of the SELECT statement. The resultant table is treated as one of source tables used in query execution. Below, we show an example of using the function to localize pattern that contains a β-strand of the length from 1 to 10 elements, optional loop up to 5 elements, α-helix of the length at least 5 elements, optional loop up to 5 elements and β-strand of any length – pattern *e(1;10),c(0;5),h(5;*), c(0;5),e(1;*)*. The pattern is searched only in proteins with the length exceeding 150 amino acids, which secondary structure was predicted (predicate *PE=4*).

```
SELECT p.protAC AS AC, p.name, s.startPos AS start, s.endPos AS end,
sequence
    AS [matched sequence], p.secondary
    FROM ProteinTbl AS p JOIN sequencePosition('secondary',
       'e(1;10),c(0;5),h(5;*),c(0;5),e(1;*)','') AS s
    ON p.id = s.proteinId
    WHERE p.name LIKE '%PE=4%' AND p.length > 150
```

The query produces results as shown in Fig. 8. It should be noted that there may be many ways how the pattern can be aligned to the protein structure from a database. The modified Smith-Waterman method returns a number of possible alignments based on a value of the internal *MPE* parameter [21]. As a result, in the table shown in Fig. 8 the same protein may appear several times with different alignment parameters.

Predicates that filter the set of rows can be defined in the WHERE clause of the SELECT statement or can be passed as the *@predicate* argument of the *sequencePosition* function.

```
AC       name               start end  matched sequence                    secondary
-------- ------------------ ----- ---- ----------------------------------- -------------------------
P75747   Protein abrB OS... 72    107  eeeeeeeeehhhhhhhhhhhhhhhhhhheeeeeeee CCCEEEEEHHHHHHHHHHHHEE...
P75747   Protein abrB OS... 222   245  eeeehhhhhhhhhhhhhhhhheee            CCCEEEEEHHHHHHHHHHHHEE...
P75747   Protein abrB OS... 136   158  eeeeehhhhhhhhhhhhcceeee             CCCEEEEEHHHHHHHHHHHHEE...
P75747   Protein abrB OS... 172   202  eeeecccccchhhhhhhhhhhhhhccceeeee    CCCEEEEEHHHHHHHHHHHHEE...
P75747   Protein abrB OS... 4     32   eeeeehhhhhhhhhhhhheeeeeeeeeee       CCCEEEEEHHHHHHHHHHHHEE...
P75747   Protein abrB OS... 22    43   eeeeeeeeeecchhhhhheeee              CCCEEEEEHHHHHHHHHHHHEE...
Q54GC8   Acyl-CoA-bindin... 172   197  eeeeeccchhhhhhhhhhcccceeee          CCCHHHHHHHHHHHHHHHHCCC...
P32104   Transcriptional... 185   212  eeeecccchhhhhhhhhhhhheeeeeee        CCCCCHHHHHHHHHHHHHHHH...
P32104   Transcriptional... 120   144  eeecccccchhhhhhcccccceeee           CCCCCHHHHHHHHHHHHHHHH...
P32104   Transcriptional... 98    123  eeeecccchhhhhhhhhhhhhhheee          CCCCCHHHHHHHHHHHHHHHH...
...
```

Fig. 8. Partial result of the search process for the given structural pattern *e(1;10),c(0;5),h(5;*),c(0;5),e(1;*)*.

However, regarding the query performance, it is better to pass them directly as the *@predicate* argument, when we call the function. This small extension forces the query processor to filter the set of proteins before creating the resultant table and before executing the Smith-Waterman method. Therefore, we do not waste time for time-consuming alignments that are not necessary in some cases. Sample query with filtering criteria specified in the function call, is shown below.

```
SELECT p.protAC AS AC, p.name, s.startPos AS start, s.endPos AS end, sequence
AS [matched sequence], p.secondary
FROM ProteinTbl AS p JOIN
        sequencePosition('secondary','e(1;10),c(0;5),h(5;*),c(0;5),e(1;*)',
        ' p.name LIKE ''%PE=4%'' AND p.length > 150') AS s
ON p.id = s.proteinId
```

6. EFFECTIVENESS AND EFFICIENCY OF PSS-SQL QUERIES

During the development of the PSS-SQL language we tested the effectiveness of the proposed solution and we have performed a set of tests in order to verify the efficiency of queries containing different patterns.

The efficacy of the PSS-SQL queries was successfully confirmed by manual comparisons of the results of *containSequence* and *sequencePosition* functions to the SSEs sequences stored in the *ProteinTbl* and segments stored in an appropriate segment table. Tests were performed for more than one hundred different SSE patterns, having different complexity, containing various numbers of segments, described precisely and rough, including SSEs of different types – defined explicitly or using wildcards.

The efficiency of the PSS-SQL queries was tested on the PC computer with the processor Intel® 3.2 GHz Core Duo and 2GB of memory. The *Proteins* database, which was used in tests, contained data describing 6 230 primary and secondary structures of proteins, as well as some additional information. Primary structures and description of proteins were downloaded from the SwissProt database [19]. Secondary structures were generated in the prediction process with the use of the Predator program [20].

The execution time of PSS-SQL queries calling the *sequencePosition* function, which localizes patterns in protein structures, takes from single seconds up to several minutes. It depends on the pattern specified in the query. In Fig. 9a we show execution times for queries containing sample patterns:

– SSE1: h(38),c(3;10),e(25;30),c(3;10),h(1;10),c(1;5),e(5;10)
– SSE2: e(4;20),c(3;10),e(4;20),c(3;10),e(15),c(3;10),e(1;10)
– SSE3: h(30;40),c(1;5),?(50;60),c(5;10),h(29),c(1;5),h(20;25)
– SSE4: h(10;20),c(1;10),h(243),c(1;10),h(5;10),c(1;10),h(10;15)
– SSE5: e(1;10),c(1;5),e(27),h(1;10),e(1;10),c(1;10),e(5;20)

The SSE1 pattern represents protein structures with the alternating α-helices and β-strands joined by loops. The SSE2 pattern represents protein structure built only with β-strands connected by loops. The SSE3 pattern consists of undefined segment of SSEs (? - wildcard). Patterns SSE4 and SSE5 have one unique region – *h(243)* and *e(27),* respectively.

We have observed, the execution time tightly depends on the uniqueness of the pattern. The more unique the pattern, the more proteins are filtered out based on the segment table, the fewer proteins are aligned by the Smith-Waterman method and the less time we need to obtain results. We can see it clearly in Fig. 9a for patterns SSE4 and SSE5, having precisely defined, unique regions *h(243)* and *e(27).* For universal patterns, for which we can find many fitting proteins or multiple alignments, we can observe longer execution times of PSS-SQL queries. In such cases, the length of the pattern influences the alignment time – for longer patterns we experience longer response times. We have not observed any dependency between the type of the SSE and the response time. However, specifying wildcards in the pattern increases the waiting period (sometimes up to several minutes). This is typical for standard SQL queries in database systems, where execution times are highly dependent on the selectivity of the queries and the number of data in a database.

Additional filtering criteria, which are commonly used in SQL queries, also decrease the execution time. In case of the *containSequence* function, additional filtering criteria can be specified only in the WHERE clause. In case of the *sequencePosition* function, they can be placed in the WHERE clause or passed as the *@predicate* parameter of the function. However, passing the criteria as parameters is better for the performance of PSS-SQL queries. The reason of this is the fact that filtering criteria in the WHERE clause are set on the resultant table of the *sequencePosition* function after it is constructed and populated. On the other hand, criteria passed as the *@predicate* parameter are set before the construction of the resultant table. In Fig. 9b we present execution times for PSS-SQL queries using the *sequencePosition* function searching the structural pattern SSE1: *h(38),c(3;10),e(25;30), c(3;10),h(1;10),c(1;5),e(5;10),* with additional filtering predicates defined as the *@predicate* parameter of the function (BUILT-IN) and in the WHERE clause:

– predicate P1: p.name like "%Homo sapiens%",
– predicate P2: p.name like "%Homo sapiens%PE=1%",
– predicate P3: p.name like "%Homo sapiens%PE=1%SV=4%",
– predicate P4: p.primary like "%NHSAAYRVDQGVLN%".

Additional predicate P1 causes the pattern to be compared only to proteins that act in *Homo sapiens* organisms. In the predicate P2 we added the condition that the candidate proteins must have the *Protein existence* attribute set to *Evidence at protein level* (*PE=1*). In predicate P3 we provided additional filter for the sequence version *SV=4*. Finally, predicate P4 sets a simple filter on the primary structure of proteins (amino acid sequence).

Analyzing the execution times of queries with additional predicates in Fig. 9b (BUILT-IN) and comparing them to the execution time of the query containing SSE1 pattern in Fig. 9a, we can notice that appropriately formulated filtering criteria significantly increase the performance of the search process and reduce the search time from several minutes even to several seconds (P3 and P4). It is also worth noting that for the analyzed pattern SSE1 we benefit from specifying additional filtering criteria as a parameter of the *sequencePosition* function. Specifying additional criteria in the WHERE clause is not so efficient in this case.

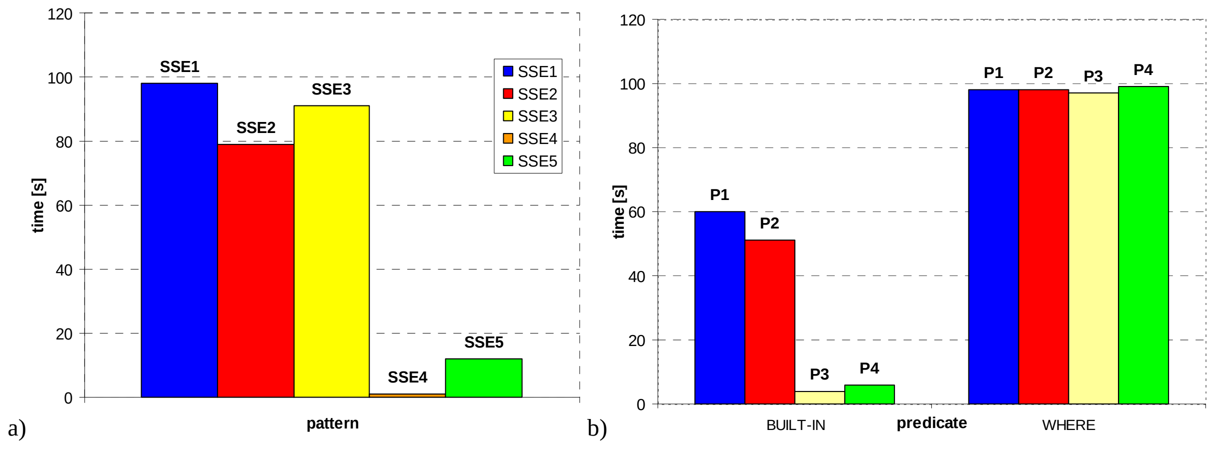

Fig. 9. Execution times of PSS-SQL queries: a) containing different patterns SSE1-SSE5, b) containing only the SSE1 pattern and various filtering predicates P1-P4 passed as a parameter (BUILT-IN) or in the WHERE clause.

7. DISCUSSION AND CONCLUDING REMARKS

PSS-SQL presented in the paper provides ready to use and easy search mechanisms that allow to find proteins having secondary structure similar to the given pattern. Comparing to existing solutions in the area (presented in section 2), the PSS-SQL has similar performance for precise patterns and can be slower for universal patterns, since they require many alignment processes. The reason of the fact is that PSS-SQL returns many possible solutions for such imprecise patterns. From this point of view, the execution times seem to be acceptable. Moreover, the syntax of the PSS-SQL is more transparent to users and more flexible in possibilities of defining query patters. The pattern defined in a query does not have to be specified strictly. Segments in the pattern can be specified as intervals and they can have undefined lengths (users can use the wildcard ‘*’ symbol). Additionally, the PSS-SQL allows to specify patterns with undefined types of the SSE (using the SSE type wildcard ‘?’ symbol) or patterns, where some SSE segments may occur optionally. Therefore, the search process has an approximate character, regarding various possible options for segments matching. Furthermore, the possibility to define patterns that include optional segments, allows users to specify gaps in a particular place.

Integrating methods of protein similarity searching with a database management system makes it easy to manipulate biological data without the need for external data mining applications. The SQL extension presented in this paper is an example of such integration. There are many advantages of the proposed extension.

First, the logic of data processing is removed from the user application and moved towards the database server. The advanced analysis of biological data is then performed while retrieving data from a database with the use of PSS-SQL queries. Therefore, the number of data returned to the user and network traffic between the server and the user application are much reduced.

Second, users familiar with the SQL syntax will easily manage to formulate PSS-SQL queries. We have designed a simple and understandable SQL extension, and in consequence, a very clear language for protein structures. This gives an advantage of the PSS-SQL language over other known solutions. However, there are many implicit operations that hide behind this simplicity and transparency, such as the alignment using the modified Smith-Waterman method, which belongs to the class of dynamic programming algorithms.

Third, as a result of PSS-SQL queries, users obtain pre-processed data. These data can then be used in further processing, e.g. users can treat results as strictly selected proteins, which meet specified criteria regarding the construction, and will be analyzed in more details. In our research, we use the presented extension in the similarity searching of protein tertiary structures. In the process, PSS-SQL queries allow us to roughly preselect proteins on the basis of their secondary structures.

BIBLIOGRAPHY

[1] EIDHAMMER I., INGE J., TAYLOR W.R., Protein Bioinformatics: An algorithmic approach to sequence and structure analysis, John Wiley & Sons, 2004.

[2] ALLEN J.P., Biophysical chemistry, Wiley-Blackwell, 2008.

[3] BRANDEN C., TOOZE J., Introduction to protein structure, Garland, 1991.

[4] DICKERSON, R.E., GEIS, I., The structure and action of proteins, 2nd ed. Benjamin/Cummings, Redwood City, Calif. Concise, 1981.

[5] CREIGHTON T.E., Proteins: Structures and molecular properties, 2nd ed. Freeman, San Francisco, 1993.

[6] GIBRAT J.F., MADEJ T., BRYANT S.H.: Surprising similarities in structure comparison, Curr Opin Struct Biol, Vol. 6(3), 1996, pp. 377–385.

[7] SHAPIRO J., BRUTLAG D., FoldMiner and LOCK 2: protein structure comparison and motif discovery on the web, Nucleic Acids Res., Vol. 32, 2004, pp. 536–41.

[8] CAN T., WANG Y.F., CTSS: a robust and efficient method for protein structure alignment based on local geometrical and biological features, Proc. 2003 IEEE Bioinformatics Conf., 2003, pp. 169–179.

[9] YANG J., Comprehensive description of protein structures using protein folding shape code, Proteins, Vol. 71(3), 2008, pp. 1497–518.

[10] BERMAN H.M., WESTBROOK J., FENG Z., GILLILAND G., BHAT T.N., WEISSIG H., et al.: The Protein Data Bank, Nucleic Acids Res., Vol. 28, 2000, pp. 235–242.

[11] DATE C.J., Introduction to database systems, (8th Edition), Addison Wesley, 2003.

[12] STEPHENS S., CHEN J.Y., THOMAS S., ODM BLAST: Sequence homology search in the RDBMS, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2004.

[13] HAMMEL L., PATEL J.M., Searching on the secondary structure of protein sequences, Proc. 28th Int. Conf. on Very Large Data Bases, Hong Kong, China, 2002, pp. 634–645.

[14] TATA S., PATEL J.M., FRIEDMAN J.S., SWAROOP A., Declarative querying for biological sequences, Proc. 22nd Int. Conf. on Data Engineering, IEEE Computer Society, 2006, pp. 87–98.

[15] WANG Y., SUNDERRAMAN R., TIAN H., A domain specific data management architecture for protein structure data, Proc. 28th IEEE EMBS Annual Int. Conf., New York City, USA, 2006, pp. 5751–5754.

[16] MURZIN A.G., BRENNER S.E., HUBBARD T., CHOTHIA C., SCOP: A structural classification of proteins database for the investigation of sequences and structures, J. Mol. Biol. Vol. 247, 1995, pp. 536–540.

[17] ORENGO C.A., MICHIE A.D., JONES S., et al., CATH – A hierarchic classification of protein domain structures, Structure, Vol. 5. No 8., 1997, pp. 1093–1108.

[18] SMITH T.F., WATERMAN M.S., Identification of common molecular subsequences, J Mol Biol, Vol. 147, 1981, pp. 195–197.

[19] APWEILER R., BAIROCH A., WU C.H., BARKER W.C., et al., UniProt: the Universal Protein knowledgebase, Nucleic Acids Res. Vol. 32 (Database issue), 2004, pp. 115–119.

[20] FRISHMAN D., ARGOS P., Incorporation of non-local interactions in protein secondary structure prediction from the amino acid sequence, Protein Eng, Vol. 9(2), 1996, pp. 133–142.

[21] WIECZOREK D., MAŁYSIAK-MROZEK B., KOZIELSKI S., MROZEK D., A method for matching sequences of protein secondary structures, Journal of Medical Informatics & Technologies, October 2010 (to be published).