J Intell Inf Syst (2016) 46:213–233
DOI 10.1007/s10844-014-0353-0

An efficient and flexible scanning of databases of protein secondary structures

with the segment index and multithreaded alignment

Dariusz Mrozek · Bartek Socha · Stanisław Kozielski · Bożena Małysiak-Mrozek

Received: 27 February 2014 / Accepted: 22 December 2014 /
Published online: 30 January 2015

Abstract Protein secondary structure describe protein construction in terms of regular spatial shapes, including alpha-helices, beta-strands, and loops, which protein amino acid chain can adopt in some of its regions. This information is supportive for protein classification, functional annotation, and 3D structure prediction. The relevance of this information and the scope of its practical applications cause the requirement for its effective storage and processing. Relational databases, widely-used in commercial systems in recent years, are one of the serious alternatives honed by years of experience, enriched with developed technologies, equipped with the declarative SQL query language, and accepted by the large community of programmers. Unfortunately, relational database management systems are not designed for efficient storage and processing of biological data, such as protein secondary structures. In this paper, we present a new search method implemented in the search engine of the PSS-SQL language. The PSS-SQL allows formulation of queries against a relational database in order to find proteins having secondary structures similar to the structural pattern specified by a user. In the paper, we will show how the search process can be accelerated by multiple scanning of the Segment Index and parallel implementation of the alignment procedure using multiple threads working on multiple-core CPUs.

Keywords Bioinformatics · Proteins · Secondary structure · Query language · Information retrieval · Parallel programming · Alignment · Structure matching · SQL · Databases

This work was supported by the European Union from the European Social Fund (grant agreement number: UDA-POKL.04.01.01-00-106/09). The work was performed using the infrastructure supported by POIG.02.03.01-24-099/13 grant: "GeCONiI-Upper Silesian Center for Computational Science and Engineering".

D. Mrozek (✉) · B. Socha · S. Kozielski · B. Małysiak-Mrozek
Institute of Informatics, Silesian University of Technology,
Akademicka 16, 44-100 Gliwice, Poland
e-mail: dariusz.mrozek@polsl.pl

1 Introduction

Structural analysis of protein molecules can be performed on one of four organizational levels, i.e. primary, secondary, tertiary, and quaternary structure (Branden and Tooze 1999). Primary structures are usually the basic source of information on the composition of a protein molecule in terms of what types of amino acids the protein is composed of and what is their order in a linear chain of the protein. Primary structure simply covers amino acid sequence of a protein (Kessel and Ben-Tal 2010). The analysis and comparison of proteins based on their primary structures is relatively simple comparing this process to the analysis and comparison of protein tertiary structures and quaternary structures (Burkowski 2008). Tertiary structures and quaternary structures, i.e. protein 3D structures, show the overall shape of proteins in three-dimensional space and require the location of all atoms of a particular molecule to be determined either by experimental methods, like X-ray crystallography or Nuclear Magnetic Resonance, or through prediction methods (Lesk 2010). The amount of protein structures in databases, such as Protein Data Bank (PDB) (Berman et al. 2000), is lagging far behind the number of amino acid sequences, e.g. in SwissProt database (Apweiler et al. 2004). Also, a simple analysis of tertiary or quaternary structures involves visualization tools, like RasMol (Sayle 1998), PyMol (Schrödinger 2010), Jmol (Jmol Homepage), or MViewer (Stanek et al. 2013). More sophisticated analyses, e.g. comparison and alignment of protein 3D structures (Ye and Godzik 2003; Eidhammer et al. 2004), require computationally complex algorithms and are usually time-consuming processes (Mrozek et al. 2014).

Secondary structures are a kind of intermediate organizational level of protein structures, a level between the simple amino acid sequence and complex 3D structure. Secondary structure, as an organizational level, describes a protein construction in terms of specific spatial shapes the amino acid chain can adopt in some of its regions. It shows how the linear chain of amino acids is formed in spiral α-helices, wavy β-strands, or loops (Fig. 1). And indeed, these three shapes, α-helices, β-strands, and loops, are main general categories of secondary structures. Secondary structure itself does not describe the location of particular atoms in 3D space. It rather reflects local hydrogen interactions between particular atoms of amino acids that are relatively close in the amino acid chain (Kessel and Ben-Tal 2010; Lesk 2010).

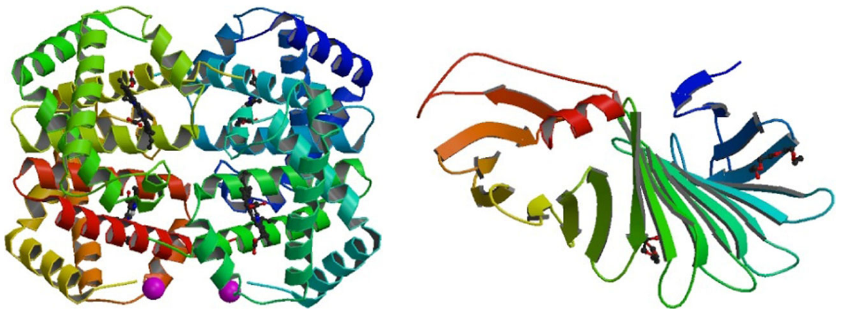


Fig. 1 Types of secondary structures: (*left*) several spiral α-helices visible in the crystal structure of human deoxyhaemoglobin (PDB ID: 2HHB) (Fermi et al. 1984), (*right*) several β-strands joined by loops in the crystal structure of OspA mutant (PDB ID: 2I5V) (Makabe et al. 2008)

The analysis of protein structures on the basis of the secondary structure types is very supportive for many processes that are important from the viewpoint of biomedicine and pharmaceutical industry, e.g. drug design. Algorithms comparing protein 3D structures and looking for structural similarities quite often make use of the secondary structure representation at the beginning, as one of the feature distinguishing one protein from the other. Secondary structures are taken into account in algorithms for protein 3D structure similarity searching, such as VAST (Gibrat et al. 1996), LOCK2 (Shapiro and Brutlag 2004), CTSS (Can and Wang 2003), CASSERT (Mrozek and Małysiak-Mrozek 2013). Also in protein 3D structure prediction by comparative modeling (Källberg et al. 2012, Yang et al. 2011), particular regions of protein structures are modeled through the adoption of particular secondary structure types of proteins that structure is already determined and deposited in a database. Secondary structure organizational level also shows what types of secondary structure a protein molecule is composed of, what is their arrangement - whether they are segregated or alternating each other. Based on the information proteins are classified by systems, such as CATH (Orengo et al. 1997) and SCOP (Murzin et al. 1995). All these examples show how important the description by means of secondary structures is.

1.1 Motivation

For scientists studying structures and functions of proteins, it is very important to collect data describing protein construction in one place and have the ability to search particular structures that satisfy given searching criteria. Consequently, this needs an appropriate representation of protein structures allowing for effective storage and searching. The problem is particularly important in the face of dynamically growing amount of biological and biomedical data in databases, such as PDB (Berman et al. 2000) or SwissProt (Apweiler et al. 2004).

At the current stage of the development of IT technologies, a well-established position in terms of collecting and managing various types of data reached relational databases (Date 2003). Relational databases collect data in tables (describing part of reality) where data are arranged in columns and rows. Modern relational databases also provide a declarative query language - SQL that allows retrieving and processing collected data. The SQL language gained a great power in processing regular data hiding details of the processing under a quite simple SELECT statement. However, processing biological data, such as protein secondary structures, by means of relational databases is hindered by several factors:

– Data describing protein structures have to be managed by database management systems (DBMSs), which work excellent in commercial uses, but 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 modern relational database systems.
– Processing of biological data must be performed by external tools and software applications, forming an additional layer in the IT system architecture, which is a disadvantage.
– Currently, results of data processing are returned in different formats, like: table-form data sets, TXT, HTML or XML files, and users must adopt them in their software applications.

– Secondary processing of the data is difficult and requires additional external tools.

In other words, modern relational database management systems (DBMSs) require some enhancements in order to deal with the data on secondary structures of proteins. A possibility of collecting protein structural data in appropriate manner and processing the data by submitting simple queries to a database would simplify the work of many researchers working in the area of protein bioinformatics.

1.2 Related works

Actually, the problem of storing biological data describing bio-polymer structures of proteins and DNA/RNA molecules in relational DBMSs and possessing appropriate methods and query language that allow processing the data has been noticed in the last decade. However, there are only a few initiatives in the world reporting this kind of solutions for various types of protein data. For example, the ODM BLAST (Stephens et al. 2004) is a successful implementation of the BLAST family of methods in the commercial Oracle database management system. ODM BLAST extends the SQL language providing appropriate functions for local alignment and similarity searching of DNA/RNA and protein amino acid sequences. ODM BLAST works fast, but in terms of protein molecules it is limited only to the primary structure. The BioSQL (BioSQL), which incorporates modules of the BioJava project (Prlić et al. 2012), provides extended capabilities by focusing on biomolecular sequences and features, their annotation, a reference taxonomy, and ontologies. However, BioSQL does not allow to process secondary structures. In Hammel and Patel (2002), authors describe their search engine and the extension to the SQL language, which allow searching on the secondary structures of protein sequences. In the solution, secondary structures are represented by segments of different types of secondary structure elements, e.g. hhhllleee. Authors have developed a dedicated search engine (Periscope) and an extension to the Oracle commercial database system - both allow to search proteins based on secondary structures. In Tata et al. (2006), 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 provides many possibilities in terms of searching based on secondary structures of proteins. In the paper (Wang et al. 2006), 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. Finally, in Małysiak-Mrozek et al. (2012) and Mrozek et al. (2010), we reported the PSS-SQL (Protein Secondary Structure - Structured Query Language) that allows to search for protein similarities on the basis of secondary structures. The search engine of the PSS-SQL utilized a single-thread alignment procedure, which we found insufficient for effective processing in the era of multi-core CPUs.

In the paper, we present a new search engine for the PSS-SQL. The search engine uses a dedicated Segment Index and a new multithreaded implementation of the alignment method in order to find protein similarities. The ideas of both techniques are presented in Section 2. In Section 4, on the basis of the results of performed experimental tests, we will prove that utilization of both techniques significantly accelerates the search process. In Section 5 we will also prove that time-consuming alignments preceded by multiple scanning of the Segment Index are competitive to existing solutions mentioned above in this section.

2 Storing and processing secondary structures in a database

Searching for protein secondary structure similarities by formulating queries in PSS-SQL requires the data describing secondary structures to be stored in a database in an appropriate format. The format should guarantee an efficient processing of the data. Moreover, searching for biological molecules, like proteins, based on the specified pattern has usually an approximate nature. This is caused by the fact that even though proteins are built up with regularly occurring building blocks, like amino acids, these building blocks form larger groups, like structural motifs or domains, and many proteins share such common regions. As a consequence, scientists usually search similarities, rather than exact matches, between protein molecules. For this reason, our new search engine relies on matching common regions and chaining the matched pairs in order to find their optimal alignment. Therefore, the search process is carried out in two main phases:

1. Multiple scanning of a dedicated Segment Index for secondary structures.
2. Alignment of found segments in order to return *k*-best solutions.

All these steps, including data preparation, creating and scanning the Segment Index, and alignment will be discussed in the following sections.

2.1 Data preparation and storing

The new search engine of the PSS-SQL uses a specific representation of protein secondary structures while storing them in a database.

Let us assume that a protein P is described by the amino acid sequence (primary structure):

$$P = \{p_i | i = 1, 2, , n \wedge p_i \in \Pi \wedge n \in \mathbb{N}\}, \quad (1)$$

where: n is the length of protein amino acid chain, i.e. the number of amino acids, and Π is a set of twenty common types of amino acids.

Secondary structure of protein P can be then described as a sequence of secondary structure elements (SSEs) related to amino acids in the protein chain:

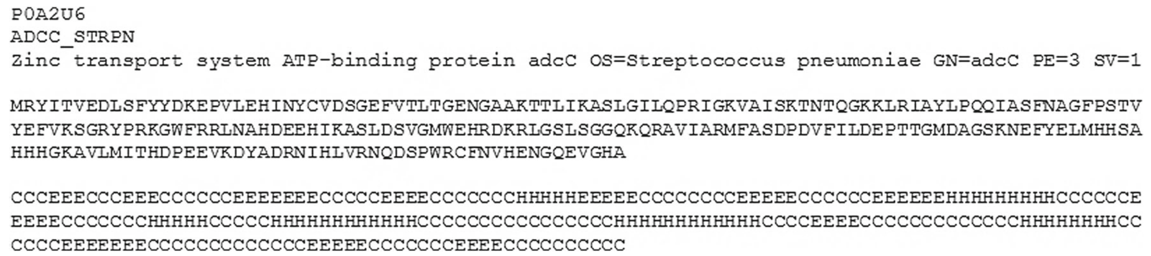
$$S = \{s_i | i = 1, 2, , n \wedge s_i \in \Sigma \wedge n \in \mathbb{N}\}, \quad (2)$$

where each element s_i corresponds to a single element p_i, and Σ is a set of secondary structure types. The set Σ may be defined in various ways. A widely-accepted definition of the set provides DSSP (Joosten et al. 2011; Kabsch and Sander 1983). The DSSP code distinguishes the following secondary structure types:

- H = alpha helix,
- B = residue in isolated beta-bridge,
- E = extended strand, which participates in beta ladder,
- G = 3-helix (3/10 helix),
- I = 5 helix (pi helix),
- T = hydrogen bonded turn,
- S = bend.

In practice, the set is often reduced to the three general types (Frishman and Argos 1996):

- H = alpha helix,
- E = beta strand (or beta sheet),
- C = 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

CCCEEECCCEEECCCCCCEEEEEEEECCCCCEEEECCCCCCCHHHHHEEEEECCCCCCCCEEEEECCCCCCEEEEEEEHHHHHHHHHCCCCCCE
EEEECCCCCCCHHHHHCCCCCHHHHHHHHHHHHCCCCCCCCCCCCCCCCHHHHHHHHHHHHCCCCEEEECCCCCCCCCCCCCHHHHHHHHCC
CCCCEEEEEEEECCCCCCCCCCCCCEEEEECCCCCCCEEEECCCCCCCCCC
```

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

An example of such a representation of protein structure is shown in Fig. 2, where we can see primary and secondary structures of a sample protein recorded as sequences. In such a way both sequences can be effectively stored in a relational database, as it is presented in Fig. 3.

2.2 Indexing of secondary structures

For the fast matching of regions that are common for the query protein structure (represented by a given query pattern) and database structures, the new search engine of the PSS-SQL uses additional data structures created in the database. A dedicated *segment table* is created for the table field storing sequences of secondary structures elements. The segment table consists of secondary structures and their lengths extracted from the sequences of SSEs, together with locations of the particular secondary structure in the molecule (identified by the residue number, Fig. 4, the *startPos* field). Then, additional Segment Index is created for the segment table. The Segment Index is a B-Tree clustered index holding on the leaf level data pages of the segment table. The idea of using the segment table and segment index is adopted from the work (Hammel and Patel 2002). The Segment Index supports preliminary filtering of protein structures that are not similar to the query pattern. During the filtering, the PSS-SQL search engine extracts the most characteristic features of the query pattern and, on the basis of the information in the index, eliminates proteins that do not meet the search criteria.

While scanning the Segment Index, the search engine of the PSS-SQL tries to match segments distinguished in the given query pattern to segments of the index. Afterward, proteins that pass the filtering process are aligned to the query pattern. This indexing technique accelerates the similarity searching.

2.3 Chaining matched pairs

In order to find optimal superposition of the query pattern on the database protein, matched pairs of protein regions are aligned. The alignment method is performed with the use of dynamic programming. When scanning a database the pairwise alignment occurs for each pair of sequences, i.e., query sequence, given by a user, and a successive candidate sequence from a database. In PSS-SQL, after performing multiple scanning of the Segment Index (MSSI), a database protein structure S^D of the length d residues is represented as a sequence of segments (Fig. 5), which can be expanded to the following form:

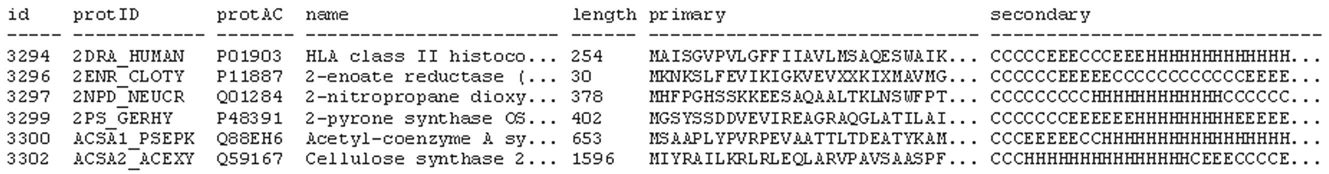
$$S^D = SSE_1^D L_1, SSE_2^D L_2, ..., SSE_n^D L_n, \tag{3}$$

where: $SSE_i^D \in \Sigma$ describes the type of secondary structure (as defined earlier), $n \leq d$ is a number of segments (secondary structures) in a database protein, $L_i \leq d$ is the length of the i^{th} segment of a database protein D.

Query protein structure S^Q, given by a user in a form of string pattern, is represented by ranges, which gives more flexibility in defining search criteria against proteins in a database:

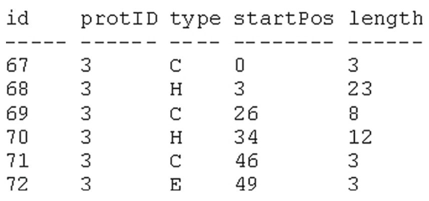
$$S^Q = SSE_1^Q(L_1; U_1), SSE_2^Q(L_2; U_2), ..., SSE_m^Q(L_m; U_m), \tag{4}$$

where: $SSE_j^Q \in \Sigma$ describes the type of secondary structure (as defined earlier), $L_j \leq U_j \leq q$ are lower and upper limits for the number of successive SSEs of the same type, q



```
id    protID       protAC  name                    length primary                          secondary
----- ------------ ------- ----------------------- ------ -------------------------------- -------------------------------
3294  2DRA_HUMAN   P01903  HLA class II histoco... 254    MAISGVPVLGFFIIAVLMSAQESWAIK...   CCCCCEEECCCEEEHHHHHHHHHHHHHH...
3296  2ENR_CLOTY   P11887  2-enoate reductase (... 30     MKNKSLFEVIKIGKVEVXXKIXMAVMG...   CCCCCCEEEEECCCCCCCCCCCCCEEEE...
3297  2NPD_NEUCR   Q01284  2-nitropropane dioxy... 378    MHFPGHSSKKEESAQAALTKLNSWFPT...   CCCCCCCCCHHHHHHHHHHHHHCCCCCC...
3299  2PS_GERHY    P48391  2-pyrone synthase OS... 402    MGSYSSDDVEVIREAGRAQGLATILAI...   CCCCCCCEEEEEEEHHHHHHHHEEEEEE...
3300  ACSA1_PSEPK  Q88EH6  Acetyl-coenzyme A sy... 653    MSAAPLYPVRPEVAATTLTDEATYKAM...   CCCEEEEEECCHHHHHHHHHHHHHHHHH...
3302  ACSA2_ACEXY  Q59167  Cellulose synthase 2... 1596   MIYRAILKRLRLEQLARVPAVSAASPF...   CCCHHHHHHHHHHHHHHHHCEEECCCCE...
```

Fig. 3 Sample relational table storing sequences of secondary structure elements SSEs (*secondary* field), amino acid sequences (*primary* field), and additional information of proteins from the Swiss-Prot database. The table (called *ProteinTbl*) will be used in sample queries presented in next sections. Secondary structures were predicted from amino acid sequences using the Predator program (Frishman and Argos 1996)



```
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

is the length of the query protein S^Q measured in residues, which is the maximal length of the string query pattern resulting from expanding the ranges of the pattern, m is a number of segments in the query pattern. Sample pattern for the query protein structure may look like as follows: `e(4;20),c(3;10),e(4;20),c(3;10),e(15),c(3;10),e(1;10).`

Additionally, the SSE_j^Q can be replaced by the wildcard symbol '?', which denotes any type of secondary structure element from Σ, and the value of the U_j can be replaced by the wildcard symbol '*', which denotes $U_j = +\infty$.

While aligning two protein structures S^Q and S^D we calculate the similarity matrix D, according to the following formulas.

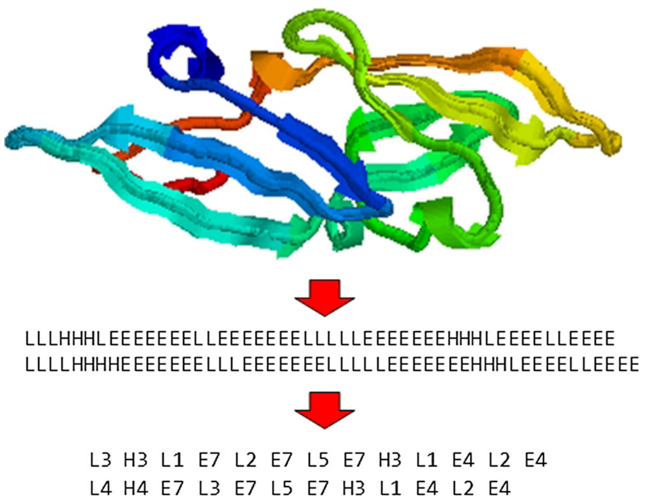
$$D_{i,0} = 0 \quad \text{for} \quad i \in [0, q], \tag{5}$$

and

$$D_{0,j} = 0 \quad \text{for} \quad j \in [0, d], \tag{6}$$

and

$$D_{i,j} = max \begin{cases} 0 \\ D_{i-1,j-1} + d_{i,j} \\ E_{i,j} \\ F_{i,j} \end{cases}, \tag{7}$$



LLLHHHLEEEEEEELLEEEEEEELLLLLEEEEEEEHHHLEEEELLEEEE
LLLLHHHHEEEEEEELLLEEEEEEELLLLLEEEEEEEHHHLEEEELLEEEE

L3 H3 L1 E7 L2 E7 L5 E7 H3 L1 E4 L2 E4
L4 H4 E7 L3 E7 L5 E7 H3 L1 E4 L2 E4

Fig. 5 Sample protein structure (PDB ID: 2EZN) (Bewley et al. 1998) reduced to the sequence of secondary structure elements (SSEs) and, after indexing, to the sequence of segments.

for $i \in [1, q]$, $j \in [1, d]$, where q, d are lengths of proteins S^Q and S^D, and $d_{i,j}$ is a matching degree between elements $SSE_i^D L_i$ and $SSE_j^Q(L_j; U_j)$ of both structures calculated by using the following formula:

$$d_{i,j} = \begin{cases} \omega_+ & \text{if} \quad SSE_i^D = SSE_j^Q \wedge L_i \geq L_j \wedge L_i \leq U_j \\ \omega_- & \text{otherwise} \end{cases}, \tag{8}$$

where: ω_+ is a matching award, and ω_- is a mismatch penalty. If the element SSE_j^Q is equal to '?', then the matching procedure ignores the condition $SSE_i^D = SSE_j^Q$. Similarly, if we assign the '*' symbol for the U_j, the procedure ignores the condition $L_i \leq U_j$.

Auxiliary matrices E and F, called gap penalty matrices, allow to calculate horizontal and vertical gap penalties with the O(1) computational complexity (as opposed to original method, where it was possible in a linear time O(n) for each direction). In the previous version of the PSS-SQL, the calculation of the current element of the matrix D required an inspection of all previously calculated elements in the same row (for a horizontal gap) and all previously calculated elements in the same column (for a vertical gap). This led to the O($q \cdot d \cdot (q+d)$) computational complexity of the entire algorithm. By using gap penalty matrices, we need only to check one previous element in a row and one previous element in a column. Such an improvement gives a significant acceleration of the alignment method by reducing general computational complexity of the algorithm to O($q \cdot d$), and taking into account the representations of both aligned proteins (3) and (4), to O($m \cdot n$). The acceleration is greater for longer sequences of secondary structure elements and larger similarity matrices D. Elements of the gap penalty matrices E and F are calculated according to the following equations:

$$E_{i,j} = max \begin{cases} E_{i-1,j} - \delta \\ D_{i-1,j} - \sigma \end{cases}, \tag{9}$$

and:

$$F_{i,j} = max \begin{cases} F_{i,j-1} - \delta \\ D_{i,j-1} - \sigma \end{cases}, \tag{10}$$

where: σ is the penalty for opening a gap in the alignment, and δ is the penalty for extending the gap, and:

$$E_{i,0} = 0 \ \text{ for } \ i \in [0, q], \quad F_{i,0} = 0 \ \text{ for } \ i \in [0, q], \tag{11}$$

$$E_{0,j} = 0 \ \text{ for } \ j \in [0, d], \quad F_{0,j} = 0 \ \text{ for } \ j \in [0, d], \tag{12}$$

The new search engine of the PSS-SQL uses the following values for matching award $\omega_+ = 4$, mismatch penalty $\omega_- = -1$, gap open penalty $\sigma = -1$, and gap extension penalty $\delta = -0.5$.

Filled similarity matrix D consists of many possible paths how two sequences of SSEs can be aligned. Backtracking from the highest scoring matrix cell and going along until a cell with score 0 is encountered allows to find the highest scoring alignment path. However, in the version of the alignment method that we have developed, we find k-best alignments by searching consecutive maxima in the similarity matrix D. This is necessary, since the pattern is usually not defined precisely, contains ranges of SSEs or undefined elements. Therefore, there can be many regions in a protein structure that fit the pattern. In the process of finding alternative alignment paths, the alignment method follows the value of the internal parameter MPE (Minimum Path End), which defines the stop criterion. We find alignment paths until the next maximum in the similarity matrix D is lower than the value of the MPE parameter. The value of the MPE depends on the specified pattern, according to the following formula.

$$MPE = (MPL \times \omega_{+}) + (NoIS \times \omega_{-}), \tag{13}$$

where: MPL is the minimum pattern length, $NoIS$ is the number of imprecise segments, i.e. segments, for which $L_i \neq U_i$. E.g. for the structural pattern `h(10;20),e(1;10),c(5),e(5;20)` containing an α-helix of the length 10 to 20 elements, a β-strand of the length 1 to 10 elements, a loop of the length 5 elements, and a β-strand of the length 5 to 20 elements, the $MPL = 21$ (10 elements of the type h, 1 element of the type e, 5 elements of the type c, and 5 elements of the type e), the $NoIS = 3$ (first, second, and fourth segment), and therefore, $MPE = 81$.

2.4 Multithreaded implementation

Standard calculation of the similarity matrix D performed by a single thread negatively affects performance of the search process or, at least, this leaves a kind of computational reserve in the era of multi-core CPUs. The new search engine for the PSS-SQL uses all processor cores that are available on the computer that hosts the database and the PSS-SQL extension.

However, this required different approach while calculating values of particular cells of the similarity matrix D. Successive cells cannot be calculated one by one, as in the original algorithm, but calculations are carried out for cells located on successive diagonals, as it is presented in Fig. 6. This is because, according to (7) each cell $D_{i,j}$ can be calculated only if there are calculated cells $D_{i-1,j-1}$, $D_{i-1,j}$ and $D_{i,j-1}$. Such an approach to the calculation of the similarity matrix is called a *wavefront*.

Moreover, in order to avoid too many synchronizations between running threads (which may lead to significant delays), the entire similarity matrix is divided to so called *areas* (Fig. 7a). These areas are parts of the similarity matrix that have a smaller size $q' \times d'$. Assuming that the entire similarity matrix has the size of $q \times d$, where q and d are lengths

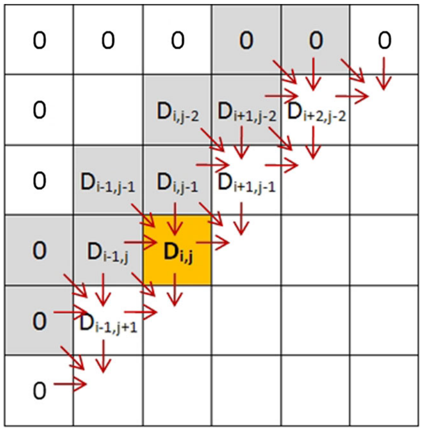


Fig. 6 Calculation of cells in the similarity matrix D by using the wavefront approach. Calculation is performed for cells at diagonals, since their values depend on previously calculated cells. Arrows show dependences of particular cells and the direction of value derivation

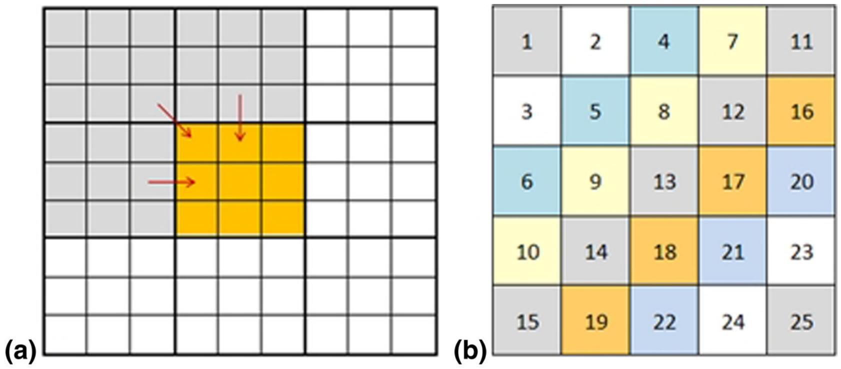


Fig. 7 Division of the similarity matrix *D* into areas. (left) Arrows show mutual dependences between areas during calculation of the matrix. (right) An order in which areas will be calculated in a sample similarity matrix

of two compared sequences of secondary structure elements, the number of areas that must be calculated is equal to:

$$n_A = \left\lceil \frac{q}{q'} \right\rceil \times \left\lceil \frac{d}{d'} \right\rceil . \tag{14}$$

For example, for the matrix D of the size 382×108 and size of the area $q' = 10$ and $d' = 10$, the $n_A = \left\lceil \frac{382}{10} \right\rceil \times \left\lceil \frac{108}{10} \right\rceil = 39 \times 11 = 429$. Areas are assigned to threads working

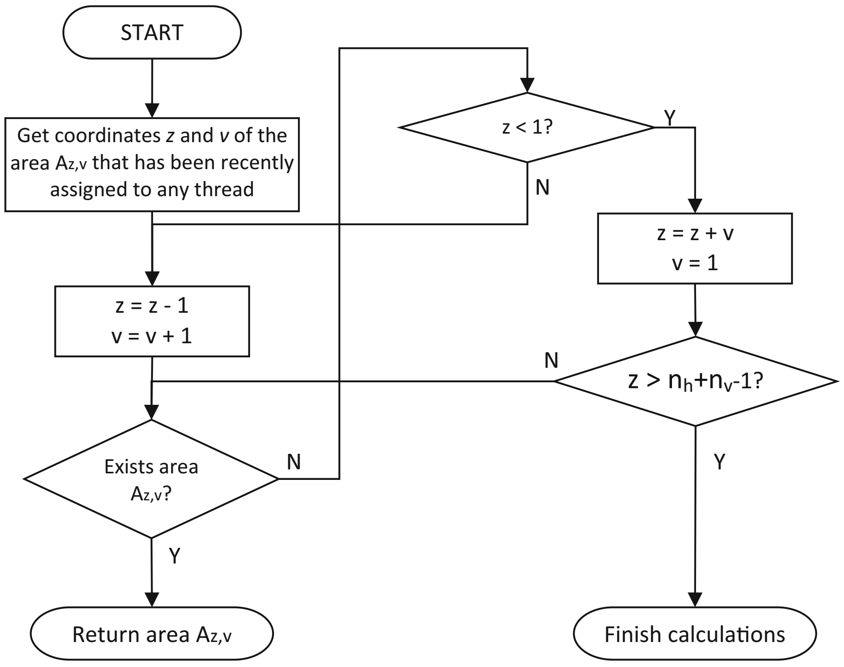


Fig. 8 Scheduling algorithm for dispatching areas to working threads in the multithreaded implementation of the alignment process (n_h, n_v - the number of areas along the horizontal and vertical edge of the similarity matrix)

in the system. Each thread is assigned to one area, which is an atomic portion of calculation for the thread. Areas can be calculated according to the same wavefront paradigm. The area $A_{z,v}$ can be calculated, if there have been calculated areas $A_{z-1,v}$ and $A_{z,v-1}$ for $z > 1$ and $v > 1$, which implicates an earlier calculation of the area $A_{z-1,v-1}$. The area $A_{1,1}$ is calculated as a first one, since there are no restrictions for calculation of the area.

In order to synchronize calculations, each area has a semaphore assigned to it. Semaphores guarantee that an area will not be calculated until the areas that it depends on have not been calculated. When all cells of an area have been calculated, the semaphore is being unlocked. Therefore, each area waits for unlocking two semaphores - for areas $A_{z-1,v}$ and $A_{z,v-1}$ for $z > 1$ and $v > 1$. The order in which areas are calculated is provided by a scheduling algorithm dispatching areas to threads (Fig. 8). When a thread completes the calculation of the current area, it is asking for another area. For example, the order of calculation of particular areas in the similarity matrix of the size 5×5 areas is presented in Fig. 7b. Such a division of the similarity matrix into areas reduces the number of tasks related to initialization of semaphores needed for synchronization purposes and reduces the synchronization time itself, which as a result, increases efficiency of the alignment algorithm.

Assuming that the value of each cell of an area is calculated at the same amount of time and taking into account the mechanism of semaphores, the CPU occupancy in particular periods of time, while calculating the similarity matrix from Fig. 7b on 4-core CPU, should look like it is shown in Fig. 9.

3 Sample queries in PSS-SQL

The PSS-SQL extension provides a set of functions and procedures for processing protein secondary structures. Three of the functions (*containSequence, sequencePosition* and *sequenceMatch*) can be effectively invoked from the SQL commands, usually the SELECT statement.

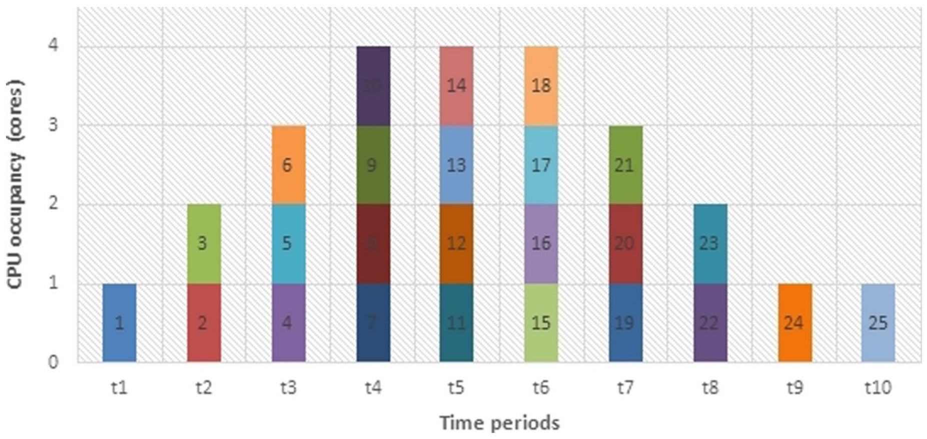


Fig. 9 Order while calculating successive areas of the similarity matrix in particular periods of time t_1, t_2, t_3... for 4 working threads. Each column corresponds to one period of time and values in columns correspond to area numbers

The *containSequence* function verifies if a particular protein or a set of database proteins contain the structural pattern specified as a query pattern. The *sequencePosition* and *sequenceMatch* functions allow to match the specified pattern to the structure of a protein or a group of database proteins. Pattern searching and matching is performed by multiple scanning of the segment index built on the segment table, followed by the alignment of the found segments. Both functions return a table containing information about the location of query pattern in the structure of each database protein. Both functions differ in the way how they are invoked in PSS-SQL queries.

Sample queries invoking both functions are shown in Listing 1. Since they return a table of values, they are nested in the FROM clause of SQL statements (mainly SELECTs, but also possible in some variants of UPDATE and DELETE statements). The use of the CROSS APPLY operator, instead of traditional JOIN, allows to avoid specifying the join condition, shortens the query syntax and, what even more important, improve performance, in the case of complex filtering conditions in the WHERE clause.

```sql
-- invoking sequenceMatch and CROSS APPLY
SELECT p.protAC AS AC, p.name, s.startPos, s.endPos,
  s.matchingSeq, p.secondary
FROM ProteinTbl AS p CROSS APPLY dbo.sequenceMatch(p.id, 'secondary',
   'e(1;10),c(0;5),h(5;6),c(0;5),e(1;10),c(5)') AS s
WHERE p.name LIKE '%Staphylococcus aureus%' AND p.length > 150
ORDER BY AC, s.startPos

-- invoking sequencePosition and standard JOIN
SELECT p.protAC AS AC, p.name, s.startPos, s.endPos, p.secondary
FROM ProteinTbl AS p JOIN dbo.sequencePosition('secondary',
   'e(1;10),c(0;5),h(5;6),c(0;5),e(1;10),c(5)',
   'p.name LIKE ''%Staphylococcus aureus%'' AND p.length > 150') AS s
  ON p.id=s.proteinId
ORDER BY AC, s.startPos
```

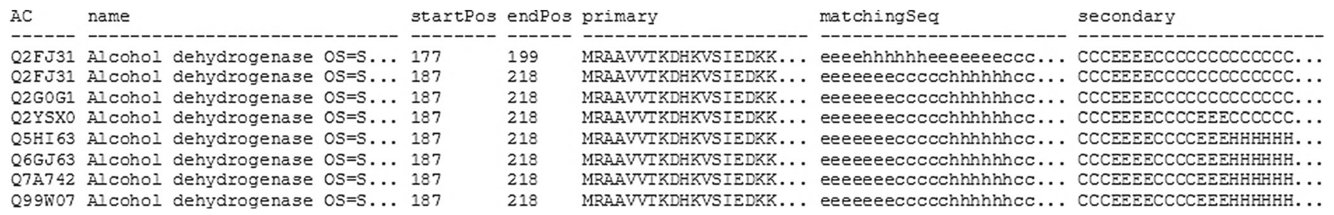
Listing 1 Sample query invoking *sequenceMatch* and *sequencePosition* table functions and returning information on proteins from *Staphylococcus aureus* having the length greater than 150 residues and containing a given secondary structure pattern.

These sample queries return Accession Numbers (AC) and names of proteins from *Staphylococcus aureus* having the length greater than 150 residues and structural region containing a β-strand of the length from 1 to 10 elements, an optional loop up to 5 elements, an α-helix of the length 5 to 6 elements, an optional loop up to 5 elements, a β-strand of the length 1 to 10 elements and a 5 element loop - pattern `e(1;10),c(0;5),h(5;6),c(0;5),e(1;10),c(5)`. Partial results of the query from Listing 1 are shown in Fig. 10.

Results of the PSS-SQL queries are originally returned in a tabular form. However, an addition of an extra FOR XML clause at the end of the SELECT statement, like in the example in Listing 2, produces results in the XML format that can be easily transformed to the HTML web page by using appropriate XSLT transformation file, and finally, published in the Internet.

```sql
SELECT p.protAC AS AC, p.name, s.startPos, s.endPos, p.[primary], s.
      matchingSeq, dbo.superimpose(s.matchingSeq, p.secondary) AS
      alignment
FROM ProteinTbl AS p CROSS APPLY dbo.sequenceMatch(p.id, 'secondary',
   'e(1;10),c(0;5),h(5;6),c(0;5),e(1;10),c(5)') AS s
WHERE p.name LIKE '%Staphylococcus aureus%'
  AND p.length > 150
ORDER BY AC, s.startPos
FOR XML RAW ('protein'), ROOT('proteins'), ELEMENTS
```

Listing 2 Sample query invoking *sequenceMatch* table function and returning results as an XML document by using the FOR XML clause.



```
AC     name                          startPos endPos primary              matchingSeq            secondary
------ ----------------------------- -------- ------ -------------------- ---------------------- ----------------------
Q2FJ31 Alcohol dehydrogenase OS=S... 177      199    MRAAVVTKDHKVSIEDKK... eeeehhhhhheeeeeeeeccc... CCCEEEECCCCCCCCCCCCC...
Q2FJ31 Alcohol dehydrogenase OS=S... 187      218    MRAAVVTKDHKVSIEDKK... eeeeeeeccccchhhhhhcc... CCCEEEECCCCCCCCCCCCC...
Q2G0G1 Alcohol dehydrogenase OS=S... 187      218    MRAAVVTKDHKVSIEDKK... eeeeeeeccccchhhhhhcc... CCCEEEECCCCCCCCCCCCC...
Q2YSX0 Alcohol dehydrogenase OS=S... 187      218    MRAAVVTKDHKVSIEDKK... eeeeeeeccccchhhhhhcc... CCCEEEECCCCEEECCCCCC...
Q5HI63 Alcohol dehydrogenase OS=S... 187      218    MRAAVVTKDHKVSIEDKK... eeeeeeeccccchhhhhhcc... CCCEEEECCCCEEEHHHHHH...
Q6GJ63 Alcohol dehydrogenase OS=S... 187      218    MRAAVVTKDHKVSIEDKK... eeeeeeeccccchhhhhhcc... CCCEEEECCCCEEEHHHHHH...
Q7A742 Alcohol dehydrogenase OS=S... 187      218    MRAAVVTKDHKVSIEDKK... eeeeeeeccccchhhhhhcc... CCCEEEECCCCEEEHHHHHH...
Q99W07 Alcohol dehydrogenase OS=S... 187      218    MRAAVVTKDHKVSIEDKK... eeeeeeeccccchhhhhhcc... CCCEEEECCCCEEEHHHHHH...
```

Fig. 10 Partial results of the sample queries from Listing 1 returned as a relational table, returned fields: *AC* - Accession Number, *name* - molecule name, *startPos*, *endPos* - position, where the pattern starts and ends in the target protein from the database, *primary* - amino acid sequence of a protein, *matchingSeq* - exact sequence of SSEs, which matches to the pattern defined in the query, *secondary* - sequence of secondary structure elements SSEs

Partial results of the query from Listing 2 are presented in Fig. 11. An additional function - *superimpose* - that was used in the presented query (Listing 2) highlights the alignment of the matched sequence and the database sequence of SSEs.

4 Efficiency of PSS-SQL

We have performed various experiments in order to test the efficiency of the new search engine for PSS-SQL query language developed as an extension to Microsoft SQL Server and Transact-SQL. Tests were performed on the Microsoft SQL Server 2012 Enterprise Edition working on nodes of the virtualized cluster controlled by the HyperV hypervisor hosted on Microsoft Windows 2008 R2 Datacenter Edition 64-bit. The host server had the following parameters: 2x Intel Xeon CPU E5620 2.40 GHz, RAM 32 GB, 3x HDD 1TB 7200 RPM. Cluster nodes were configured to use 4 CPU cores and 4GB RAM per node, and worked under the Microsoft Windows 2008 R2 Enterprise Edition 64-bit operating system.

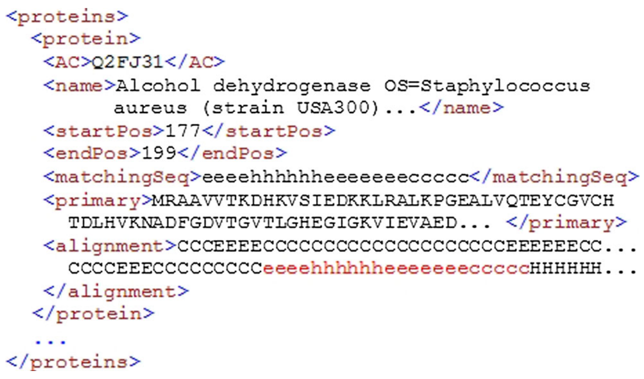
Most of our tests were performed on the database storing 6 360 protein structures. However, in order to compare our search engine to one of the competitive solutions, we performed some tests on the database storing 248 375 protein structures.

During our experiments we measured execution times for various query patterns. The query patterns were passed as a parameter of the *sequencePosition* function. Tests were performed for queries containing the following sample patterns:

– SSE1: `e(4;20),c(3;10),e(4;20),c(3;10),e(15),c(3;10),e(1;10)`
– SSE2: `h(30;40),c(1;5),?(50;60),c(5;10),h(29),c(1;5),h(20;25)`
– SSE3: `h(10;20),c(1;10),h(243),c(1;10),h(5;10),c(1;10),h(10;15)`
– SSE4: `e(1;10),c(1;5),e(27),h(1;10),e(1;10),c(1;10),e(5;20)`
– SSE5: `e(5;20),h(2;5),c(2;40),?(1;30),e(5;*)`

Pattern SSE1 represents protein structure built only with β-strands connected by loops. Pattern SSE2 consists of several α-helices connected by loops and one undefined segment of SSEs ('?' wildcard symbol). Patterns SSE3 and SSE4 have regions that are unique in the database, i.e. `h(243)` in pattern SSE3 and `e(27)` in pattern SSE4. Pattern SSE5 has a wildcard symbol '*' for undetermined length, which slows down the search process.

In order to verify the influence of particular acceleration techniques on the execution times, tests were carried out for the PSS-SQL in three variants:



```
<proteins>
  <protein>
   <AC>Q2FJ31</AC>
   <name>Alcohol dehydrogenase OS=Staphylococcus
         aureus (strain USA300)...</name>
   <startPos>177</startPos>
   <endPos>199</endPos>
   <matchingSeq>eeeehhhhhheeeeeeeeccccc</matchingSeq>
   <primary>MRAAVVTKDHKVSIEDKKLRALKPGEALVQTEYCGVCH
     TDLHVKNADFGDVTGVTLGHEGIGKVIEVAED... </primary>
   <alignment>CCCEEEECCCCCCCCCCCCCCCCCCCEEEEEECC...
     CCCCEEECCCCCCCCCeeeehhhhhheeeeeeeecccccHHHHHH...
   </alignment>
  </protein>
  ...
</proteins>
```

Fig. 11 Partial results of the query from Listing 2 returned as an XML file

- without multithreading (–MT, on an old search engine),
- with multithreading, but without multiple scanning of the Segment Index (+MT–MSSI, on a new search engine),
- with multithreading and with multiple scanning of the Segment Index (+MT+MSSI, on a new search engine).

Results of the tests presented in Fig. 12 prove that the performance of +MT–MSSI variant is higher, and in case of SSE1 and SSE2 even much higher, than –MT variant (implemented in the previous search engine of the PSS-SQL). For +MT+MSSI we can see additional improvement of the performance. It is difficult to estimate the overall acceleration, because it tightly depends on the uniqueness of the pattern. The more unique the pattern is, the more proteins are filtered out based on the Segment Index, the fewer proteins are aligned and the less time we need to obtain results. We can see it clearly in Fig. 12 for patterns SSE3 and SSE4 that have precisely defined, unique regions `h(243)` and `e(27)`. For universal patterns, like SSE1 and SSE2, for which we can find many fitting proteins or multiple alignments, we can observe longer execution times. In such cases, the parallelization and multiple scanning of the Segment Index start playing a more significant role. In these cases, the length of the pattern influences the alignment time - for longer patterns we experienced longer response times. We have not observed any dependency between the type of the SSE and the response time.

However, specifying wildcards in the query pattern increases the waiting period, which is visible for the pattern SSE5 (Fig. 13). In Fig. 13 for the pattern SSE5 we can also see how beneficial the use of the MSSI technique can be. In this particular case, the execution time was reduced from 920 seconds in –MT (old search engine), and 550 seconds in +MT–MSSI, to 15 seconds in +MT+MSSI, which gives 61.33-fold speed up over the –MT variant and 36.67-fold speed up over the +MT–MSSI variant.

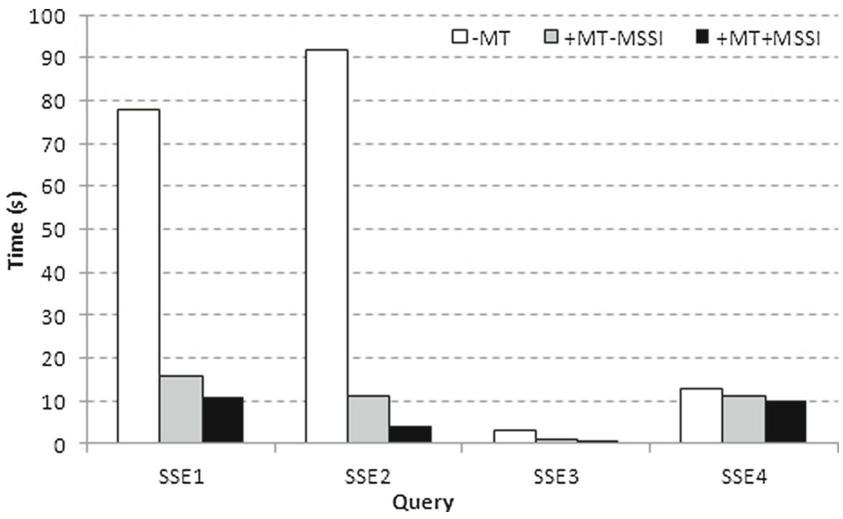


Fig. 12 Execution time for various query patterns SSE1-SSE4 and for three variants of the PSS-SQL language: without multithreading (–MT, old search engine), with multithreading, but without multiple scanning of the Segment Index (+MT–MSSI, new search engine), with multithreading and with multiple scanning of the Segment Index (+MT+MSSI, new search engine)

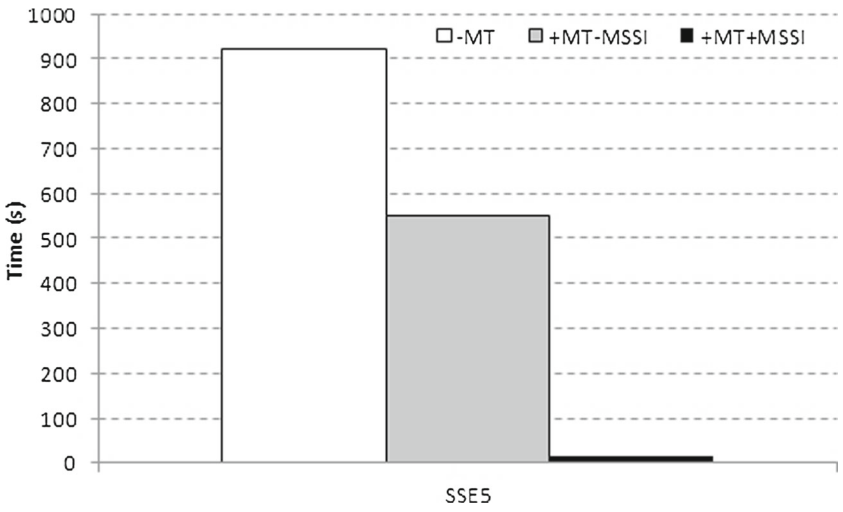


Fig. 13 Execution time for query pattern SSE5 for three variants of the PSS-SQL language: without multi-threading (–MT, old search engine), with multithreading, but without multiple scanning of the Segment Index (+MT–MSSI, new search engine), with multithreading and with multiple scanning of the Segment Index (+MT+MSSI, new search engine)

5 Discussion

PSS-SQL language with the new search engine complements existing relational database management systems, which are not designed to process biological data, such as protein secondary structures. By extending the standard SELECT, UPDATE and DELETE statements of the SQL language, the PSS-SQL provides a declarative method for retrieving, modifying and deleting records. Records that satisfy the criteria given by a user can be returned in a table-like form or as an XML document, which is easy to display as a web page. In such a way, the PSS-SQL extension to RDBMS provides a kind of domain specific language for processing protein secondary structures. This is especially important for relational database designers, wide group of biological data analysts and bioinformaticians.

The PSS-SQL language with the new search engine can be used for the fast classification of proteins based on their secondary structures. For example, systems such as SCOP (Murzin et al. 1995) and CATH (Orengo et al. 1997) make use of the secondary structure description of protein structures in order to classify proteins into classes and families. PSS-SQL can be also supportive in protein 3D structure prediction by homology modeling, where appropriate structure profile can be found based on primary and secondary structure and the secondary structure can be superimposed on the protein of the unknown 3D structure before performing a free energy minimization.

Comparing the new search engine of the PSS-SQL to other approaches presented in Section 1.2, we can notice that all variants of the PSS-SQL extend the syntax of the SQL. This makes the PSS-SQL similar to PiQL (Tata et al. 2006), rather than to ProteinQL (Wang et al. 2006). ProteinQL was developed for the Object-Oriented Database and relies on its own domain specific database and dedicated ProteinQL interpreter and translator. As opposed to ProteinQL, both PiQL and PSS-SQL extend capabilities of Relational Database Management Systems (RDBMS). They extend the syntax of the SQL language by providing additional functions that can be nested in particular clauses of the SQL commands. However, the form of queries provided by users is different. PiQL accepts query patterns in a full form that is similar to BLAST (Altschul et al. 1990) - a tool used for fast local matching of bio-molecular sequences of DNA and proteins. Query patterns provided in PSS-SQL are similar to those presented by Hammel and Patel (2002). 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. Both languages allow specifying query patterns with undefined types of the SSE or patterns, where some SSE segments may occur optionally. Therefore, the search process has an approximate nature, regarding various possible options for segment matching. The possibility of defining patterns that include optional segments allows users to specify gaps in a particular place.

In the new search engine of the PSS-SQL we also used the method of scanning the Segment Index in order to accelerate the search process. The method was adopted from the work of Hammel and Patel (2002). However, after multiple scans of the Segment Index Hammel and Patel used sort-merge join operations in order to join segments from the same candidate proteins and decide, whether they meet specified query conditions or not. The novelty of the new search engine of the PSS-SQL lies in the alignment of the found segments. Alignment implemented in PSS-SQL gives the unique possibility of finding many matches for the same database protein and returning *k*-best matches, matches that in some particular cases can be separated by gaps. These are not the gaps defined by a user and specified by an optional segment, but the gaps providing better alignment of particular regions. This type of matching is typical for similarity searching between biomolecular sequences, such as DNA/RNA sequences or amino acid sequences. Presented approach extends the spectrum of searching and guarantees the optimality of the results according to assumed scoring system.

Despite the fact that our solution uses the alignment procedure, which is computationally complex, it gained quite a good performance. We have compared the efficiency of the new search engine of the PSS-SQL (+MT+MSSI variant) and language presented by Hammel and Patel for single-predicate exact match queries with various selectivity (between 0.3 % and 6 %) using the database storing 248 375 proteins (515 MB for *ProteinTbl* used in examples, 254 MB for segment table storing 11 986 962 segments). The new search engine of the PSS-SQL was on average 5.14 faster than *Comm-Seg* implementation, 3.28 faster than *Comm-CSP* implementation, both implemented on a commercial ORDBMS, and 1.84 faster than *ISS-MISS(1)* implementation on Periscope/SQ (Hammel and Patel 2002). This proves, that our new search engine compensates the efficiency loss caused by alignment procedure by using the Segment Index. In such a way, the PSS-SQL joins wide capabilities of the alignment process (possible gaps, mismatches, and many solutions), provides optimality and quality of results, and guarantees efficiency of scanning databases of secondary structures.

6 Summary

Integrating methods of protein secondary structure similarity searching with database management systems provides an easy way for manipulation of biological data without the necessity of using external data mining applications. The PSS-SQL extension presented in this paper is a successful example of such integration. PSS-SQL is certainly a good option for biological and biomedical data analysts who want to process their data on the server side. This has many advantages that are typical for such a processing in the client-server architecture. Entire logic of data processing is performed on the database server, which reduces the load on the user's computer. Therefore, data exploration is performed while retrieving data from a database. Moreover, the number of data returned to the user, and the network traffic between the server and the user application, are much reduced.

The new search engine of the PSS-SQL with implemented multithreaded alignment procedure allows to utilize the whole capable computing power more efficiently. The search engine adapts to the number of processing units possessed by the server, which hosts the database management system, and to the number of cores used by the database system. This results in better performance while scanning huge databases of protein secondary structures.

References

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*, 403–10.

Apweiler, R., Bairoch, A., Wu, C.H., et al. (2004). Uniprot: the Universal Protein knowledgebase. *Nucleic Acids Research*, *32*(Database issue), D115–9.

Berman, H. et al. (2000). The Protein Data Bank. *Nucleic Acids Research*, *28*, 235–242.

Bewley, C.A., Gustafson, K.R., Boyd, M.R., Covell, D.G., Bax, A., Clore, G.M., Gronenborn, A.M. (1998). Solution structure of cyanovirin-N, a potent HIV-inactivating protein. *Natural Structural Biology*, *5*(7), 571–8.

BioSQL. http://biosql.org/.

Branden, C., & Tooze, J. (1999). *Introduction to Protein Structure*, 2nd ed: Garland Science.

Burkowski, F. (2008). *Structural Bioinformatics: An Algorithmic Approach*, 1st ed: Chapman and Hall/CRC.

Can, T., & Wang, Y. (2003). CTSS: A robust and efficient method for protein structure alignment based on local geometrical and biological features. In *Proceedings of the 2003 IEEE Bioinformatics Conference (CSB 2003)* (pp. 169–179).

Date, C. (2003). *An introduction to database systems*, 8th edn. USA: Addison-Wesley.

Eidhammer, I., Inge, J., Taylor, W.R. (2004). *Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis*: John Wiley & Sons.

Fermi, G., Perutz, M.F., Shaanan, B., Fourme, R. (1984). The crystal structure of human deoxyhaemoglobin at 1.74 A resolution. *Journal of Molecular Biology*, *175*, 159–174.

Frishman, D., & Argos, P. (1996). Incorporation of non-local interactions in protein secondary structure prediction from the amino acid sequence. *Protein Engineering*, *9*(2), 133–142.

Gibrat, J., Madej, T., Bryant, S. (1996). Surprising similarities in structure comparison. *Current Opinion in Structural Biology*, *6*(3), 377–385.

Hammel, L., & Patel, J.M. (2002). Searching on the secondary structure of protein sequences. In *Proceedings 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002* (pp. 634–645).

Jmol Homepage. Jmol: an open-source Java viewer for chemical structures in 3D. http://www.jmol.org.

Joosten, R.P., Te Beek, T.A.H., Krieger, E., Hekkelman, M.L., et al. (2011). A series of PDB related databases for everyday needs. *Nucleic Acid Research*, *39*(Database issue), D411–D419.

Kabsch, W., & Sander, C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, *22*, 2577–2637.

Källberg, M., Wang, H., Wang, S., Peng, J., Wang, Z., Lu, H., Xu, J. (2012). Template-based protein structure modeling using the RaptorX web server. *Nature Protocols*, *7*, 1511–1522.

Kessel, A., & Ben-Tal, N. (2010). *Introduction to Proteins: Structure, Function, and Motion*, 1ed: Chapman & Hall/CRC Mathematical & Computational Biology, CRC Press.

Lesk, A.M. (2010). *Introduction to Protein Science: Architecture, Function, and Genomics*, 2ed. USA: Oxford University Press.

Makabe, K., Biancalana, M., Yan, S., Tereshko, V., Gawlak, G., Miller-Auer, H., Meredith, S.C., Koide, S. (2008). High-resolution structure of a self-assembly-competent form of a hydrophobic peptide captured in a soluble beta-sheet scaffold. *Journal of Molecular Biology*, *378*, 459–467.

Małysiak-Mrozek, B., Kozielski, S., Mrozek, D. (2012). Server-Side Query Language for Protein Structure Similarity Searching. In *In: Human - Computer Systems Interaction: Backgrounds and Applications. Advances in Intelligent and Soft Computing*, (Vol. 2 pp. 395–415). Berlin Heidelberg: Springer.

Mrozek, D., Brożek, M., Małysiak-Mrozek, B. (2014). Parallel implementation of 3D protein structure similarity searches using a GPU and the CUDA. *Journal of Molecular Modeling*, *20*, 2067.

Mrozek, D., & Małysiak-Mrozek, B. (2013). CASSERT: A Two-Phase Alignment Algorithm for Matching 3D Structures of Proteins. In Kwiecień, A., Gaj, P., Stera, P. (Eds.) *Proceedings of 22nd International Conference on Computer Networks, Communications in Computer and Information* (Vol. 370, pp. 334–343): Springer-Verlag, CCIS.

Mrozek, D., Wieczorek, D., Małysiak-Mrozek, B., Kozielski, S. (2010). PSS-SQL: Protein Secondary Structure - Structured Query Language. Proceedings of 32th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS 2010 (pp. 1073–1076). Argentina: Buenos Aires.

Murzin, A.G., Brenner, S.E., Hubbard, T., Chothia, C. (1995). SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, *247*, 536–540.

Orengo, C.A., Michie, A.D., Jones, S., Jones, D.T., et al. (1997). CATH - A hierarchic classification of protein domain structures. *Structure*, *5*(8), 1093–1108.

Prlić, A., Yates, A., Bliven, S.E., Rose, P.W., et al. (2012). BioJava: an open-source framework for bioinformatics in 2012. *Bioinformatics*, *28*, 2693–2695.

Schrödinger, L.L.C. (2010). The PyMOL molecular graphics system, version 1.3r1 . *PyMOL The PyMOL Molecular Graphics System, Version 1.3*: Schrödinger, LLC. http://www.pymol.org.

Sayle, R. (1998). RasMol, Molecular Graphics Visualization Tool. Biomolecular Structures Group, Glaxo Welcome Research & Development, Stevenage, Hartfordshire, 5/02/2013. http://www.umass.edu/microbio/rasmol/.

Shapiro, J., & Brutlag, D. (2004). FoldMiner and LOCK2: protein structure comparison and motif discovery on the web. *Nucleic Acids Research*, *32*, 536–41.

Stanek, D., Mrozek, D., Małysiak-Mrozek, B. (2013). MViewer: Visualization of protein molecular structures stored in the PDB, mmCIF and PDBML data formats. In Kwiecień, A., Gaj, P., Stera, P. (Eds.) *CN 2013* (Vol. 370, pp. 323–333): CCIS.

Stephens, S., Chen, J.Y., Thomas, S.h. (2004). ODM BLAST: Sequence homology search in the RDBMS. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering.

Tata, S., Patel, J.M., Friedman, J.S., Swaroop, A. (2006). Declarative querying for biological sequences. *Proceedings 22nd International Conference on Data Engineering, IEEE Computer Society*, 87–98.

Wang, Y., Sunderraman, R., Tian, H. (2006). A domain specific data management architecture for protein structure data. Proceedings 28th IEEE EMBS Annual Int. Conf., New York City, USA, 2006, pp 5751–5754.

Yang, Y., Faraggi, E., Zhao, H., Zhou, Y. (2011). Improving protein fold recognition and template-based modeling by employing probabilistic-based matching between predicted one-dimensional structural properties of the query and corresponding native properties of templates. *Bioinformatics*, *27*, 2076–82.

Ye, Y., & Godzik, A. (2003). Flexible structure alignment by chaining aligned fragment pairs allowing twists. *Bioinformatics*, *19*(2), 246–255.