Supplementary information for “OverProt: secondary structure consensus for protein families”

Adam Midlik[1,2], Ivana Hutařová Vařeková[1,2,3], Jan Hutař[1,2], Aliaksei Chareshneu[1,2], Karel Berka[4,*], and Radka Svobodová[1,2,*]

[1]CEITEC – Central European Institute of Technology, Masaryk University, Kamenice 5, 625 00 Brno, Czech Republic

[2]National Centre for Biomolecular Research, Faculty of Science, Masaryk University, Kamenice 5, 625 00 Brno, Czech Republic

[3]Faculty of Informatics, Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic

[4]Department of Physical Chemistry, Regional Centre of Advanced Technologies and Materials, Faculty of Science, Palacký University, 17. listopadu 1192/12, 771 46 Olomouc, Czech Republic

[*]To whom correspondence should be addressed.

Table of contents

1 Introduction

OverProt is a tool for construction and visualization of the secondary structure consensus for protein families. The consensus produced by OverProt can be used as a template for annotation of secondary structure elements by SecStrAnnotator.

OverProt consists of three main parts: the main algorithm **OverProt Core** constructs the secondary structure consensus, **OverProt Viewer** visualizes the consensus, and **OverProt Server** presents the results on the web and allows user-defined computations.

2 Terminology

- **Protein structure** – a set of atoms with assigned 3D coordinates. A structure consists of one ore more **chains**. A chain is a sequence of **residues**, which consist of the individual **atoms**. OverProt works with structures in **mmCIF format**. Structures deposited in the PDB [cite] are referenced by their PDB ID (e.g. `1tqn`). OverProt follows the *label** numbering scheme when referencing chains and residues within a structure (i.e. items `label_asym_id` and `label_seq_id` in the mmCIF file) – this is sometimes different from the *auth** numbering scheme.

- **Protein domain** – a part of protein structure which may be a whole chain or a range (ranges) of residues in a chain. A domain is defined by the structure identifier, chain identifier, and one or more ranges of residues, e.g. `1tqn,A,7:478` or `1n26,A,2:9,94:192`. Residue ranges include the start and end residue (e.g. `5:8` means residues 5, 6, 7, 8).

- **Protein family** – a set of protein domains with reasonable structural similarity. The set can be provided by the user or it can be defined based on the CATH database [cite], in which case the family (*CATH superfamily*) is identified by its CATH ID (e.g. `1.10.630.10`) and domains are identified by CATH domain ID (e.g. `1tqnA00`).

- **Secondary structure element (SSE)** – a section of a protein chain with some secondary structure pattern. OverProt focuses on two types of SSEs – **helices** (H) and **β-strands** (E). Each SSE within a protein structure can be identified by its chain identifier, start (index of its first residue), end (index of its last residue), and type. For comparing SSEs, it is convenient to simplify an SSE to a line segment (i.e. 3D coordinates of the start and end point).
  The term β-connectivity refers to the way in which the strands are connected: a **β-ladder** is a connection of two strands and can be either parallel or antiparallel; a **β-sheet** is a set of strands which are connected by β-ladders.
  This model is kept as simple as possible (different helix types (α, 3_{10}, π) are not distinguished; other SSE type (loops, turns) are not taken into account). Secondary structure assignment (i.e. detection of SSEs) is performed by **SecStrAnnotator**, more details can be found in its original paper [cite].

- **Consensus SSE** – a set of equivalent SSEs from different family members. The **occurrence** of a consensus SSE is the number of domains that contain it, divided by the total number of domains in the family. (e.g. consensus helix X = {helix H1 in domain1, helix

H3 in domain2, helix H3 in domain3}; domain4 contains no helix X; the occurrence is 3/4 = 0.75 or 75%).

- **Secondary structure consensus** – a set of consensus SSEs together with their order and β-connectivity.

3 Methods – OverProt Core

OverProt Core is an algorithm that constructs the secondary structure consensus for a given protein family. The algorithm proceeds in a number of steps, combined in `overprot.py`:

3.1 Preparation

- Download the list of domains for the family (if not already given by `--domains`) from PDBe API (`https://www.ebi.ac.uk/pdbe/api/mappings/{family_id}`).

- Select sample: If `sample_size` is different from `all`, select a random subset of the domain list.
  The family may contain multiple domains from the same PDB entry. If `[sample_selection]unique_pdb` is `True`, then these are treated as duplicates and only one of them is selected (the first in alphabetical order).

- Download structures: The structures of listed domains are downloaded in mmCIF format, domains are cut out from the structures and saved in separate files. The sources of structures are given by `--structure_source` and `[download]structure_sources`. Structures are also converted to PDB format for some later steps (MAPSCI). The download step is performed by `StructureCutter` written in C#.

3.2 Structural alignment

Multiple structure alignment is performed in 2 steps:

- Program MAPSCI [cite Ilinkin et al., 2010] is used to calculate a consensus structure (-> `mapsci/consensus.cif`). For performance reasons, at most 100 domains are selected for this calculation (in a quasi-random way, i.e. for the same family selects each time the same subset).
  To reduce indeterminism and ease later visualization, the consensus structure is centered to the origin (0, 0, 0), rotated so that its PCA components are aligned to the XYZ axes (“the structure is laid flat”), and flipped in a consistent way (roughly so that the chain goes left-top to right-bottom and its ends are more in front).
  In general, MAPSCI produces a reasonable consensus structure, but the alignment of the individual domains is often poor – therefore the following re-alignment step is necessary.

- In the realignment step, all domains are structurally aligned onto the consensus structure via `cealign` algorithm [cite Shindyalov and Bourne, 1998] provided in PyMOL module [cite]. In rare cases the cealign fail (when the domain is too short) – in such cases a simple alignment algorithm DumbAlign is used instead (theretically inefficient and inaccurate, but sufficient for these very short domains).

3.3 Secondary structure assignment

The SSEs in each domain are detected by SecStrAnnotator [cite Midlik et al., 2019, 2021] (options --onlyssa --verbose --batch).

3.4 Guide tree

The domains are clustered by agglomerative clustering to produce a guide tree. During this step, each domain is first converted into a **weighted structure** (a sequence of C-alpha coordinates with individual weight for each point). The two most similar weighted structures are then merged into a new weighted structure, and this is repeated until we end up with a single weighted structure, corresponding to the tree root.

This agglomerative algorithm can be expressed by the following pseudocode:

```
Workset = { a weighted structure for each input domain }
while |Workset| > 1:
    A, B = two nearest weighted structures in Workset
    C = merge(A, B)
    Children[C] = {A, B}
    Workset = Workset - {A, B} ∪ {C}
```

At the end, Workset will only contain one weighted structure, which is the tree root. The topology of the tree will be defined by Children.

Example: 5 structures were initially in Workset. B+C were merged into BC, then D+E into DE, then A+BC into ABC, and finally ABC+DE into ABCDE. The resulting tree is:

```
A────────────┐
             │
B────┐       ├ABC──────┐
     ├BC─────┘         │
C────┘                 ├ABCDE
                       │
D────────┐             │
         ├DE───────────┘
E────────┘
```

Details of the algorithm are described in Appendix (distance function D^*, which determines the nearest weighted structures, operation merge).

3.5 Merging

This is the core of the consensus generation algorithm. On the input, we have a set of k protein domains. Each domain is simplified to a sequence of SSEs (defined by their type, line segment, etc.). The required output is a clustering of all input SSEs. However, the clustering must fulfil these constraints:

1. Each cluster can contain only elements of the same type (only helices or only strands).
2. A cluster must not contain more than one element from the same protein domain.
3. There must be a partial ordering of the clusters.

The third constraint can be formalized:

- SSE x precedes SSE y (written $x \rightarrow y$) iff they are in the same protein domain and x appears before y.
- Cluster A precedes cluster B ($A \rightarrow B$) iff there exist SSEs $x \in A, y \in B$ such that $x \rightarrow y$.
- There must be no sequence of clusters $A, B...$, such that $A \rightarrow B \rightarrow ... \rightarrow A$.

TODO figures (from presentations/posters) - GM pr16

In the merging step, the guide tree leaves are first populated with SSEs of their respective domains. In each internal node, the SSEs from the two children nodes are matched together and merged. The root then contains the consensus SSEs of the whole family. The matching and merging of two SSE graphs is in principle similar to matching (aligning) and merging of two weighted structures. The best matching is also found by dynamic programming. However, it is more complicated here, because 1) clusters of different type cannot be matched (this can cause the branching in the resulting graph), and 2) the dynamic programming algorithm is not as straightforward for matching DAGs as it is for matching sequences.

TODO say what is implemented in which function

3.6 Annotation

This is an optional step, in which the generated SSE consensus is used as annotation template for SecStrAnnotator and all family members are annotated. Before the annotation, the SSEs with low occurrence (< 5%) are removed, which dramatically reduces the running time of SecStrAnnotator. Options for SecStrAnnotator: `--ssa file --align none --metrictype 3 --fallback 30 --unannotated`.

3.7 Visualization

The generated SSE consensus is visualized by several SVG diagrams with different settings and `diagram.json` file is produced, which will be used for interactive visualization be OverProt Viewer. A PyMOL session (.pse) is created, with the structure consensus from MAPSCI (shown as ribbon) and the SSE consensus (shown as flat-end cylinders (helices) and round-end cylinders (strands)). A PNG image is also rendered from the session. A session with all domains and their SSEs is generated if `[visualization]create_multi_session` is `True` (very slow, not recommended for larger families).

3.8 Multi-family mode

Multiple families can be processed in parallel using `overprot_multifamily.py`.

4 Interactive visualization by OverProt Viewer

OverProt Viewer is a web component for interactive visualization of the SSE consensus. Its input is the preprocessed `diagram.json` file. It is implemented in TypeScript with D3.js.

5 Data computation for OverProt Server

OverProt Server serves precomputed SSE consensuses (database) and runs the OverProt Core algorithm for user-defined sets of domains (jobs).

The database is constructed in this way:

- Retrieve the current list of families from CATH (ftp://orengoftp.biochem.ucl.ac.uk /cath/releases/latest-release/cath-classification-data/cath-domain-list.txt). (This file also contains the domain definitions, but they are in an incompatible numbering scheme (*auth**), therefore they are not used). This is currently 6631 families, out of which 64 are empty families (December 2021).
- Retrieve the domain lists for each family, including chains and residue ranges, from PDBe API (https://www.ebi.ac.uk/pdbe/api/mappings/{family_id}). This is currently over 470k domains (December 2021).
- Remove duplicates (i.e. multiple domains from the same PDB entry). This is currently over 200k domains (December 2021).
- Apply the OverProt Core algorithm to each family.

The whole process is realized by:

```
python overprot_multifamily.py --download_family_list_by_size \
--config working_scripts/overprot-config-overprotserverdb.ini \
--collect xxx all $UPDATE_DIRECTORY
```

The database is updated weekly.

6 Appendix

6.1 Computation of distance function for two weighted structures

A weighted structure A is a tuple $(n^A, \mathrm{R}^A, \mathrm{W}^A, k^A)$ where n^A is the length of the weighted structure (number of points), R^A is the matrix of their coordinates ($n^A \times 3$), W^A is the vector of their relative weights $\in (0, 1]$, and k^A is the absolute weight of A (scalar). Example of a weighted structure:

$$n^A = 4 \quad \mathrm{R}^A = \begin{bmatrix} -1.1 & -2.9 & 0.1 & 0.4 \\ 0.0 & 1.1 & 0.9 & -2.7 \\ 5.2 & 2.1 & 0.0 & 0.8 \end{bmatrix} \quad \mathrm{W}^A = \begin{bmatrix} 1 & 0.5 & 0.8 & 1 \end{bmatrix} \quad k^A = 10$$

r_i^A and w_i^A will refer to i-th column of R^A and W^A.

TODO nice format (matrices and vectors in bold)

A protein domain can be converted into a weighted structure as follows: n is the number of residues, r_i^A are the coordinates of the C-alpha atom of i-th residue, w_i^A is 1, and k^A is 1.

Distance function d is defined for two weighted points:

$$d((r_i^A, w_i^A), (r_j^B, w_j^B)) = \left(1 - e^{-\|r_i^A - r_j^B\|/R_0}\right) \cdot \min\{w_i^A, w_j^B\} + \frac{1}{2}|w_i^A - w_j^B|$$

In case that one of the weighted points is undefined ($\perp$), d is still defined:

$$d((r_i^A, w_i^A), \perp) = \frac{1}{2}w_i^A \qquad d(\perp, (r_j^B, w_j^B)) = \frac{1}{2}w_j^B$$

(d is not the Euclidean distance of the two points. $d \in [0, 1)$.)

An alignment of two weighted structures A, B is a sequence of pairs $[(p_1, q_1), (p_2, q_2), ..., (p_n, q_n)]$ (for better reading written as a matrix $\begin{bmatrix} p_1 & p_2 & ... & p_n \\ q_1 & q_2 & ... & q_n \end{bmatrix}$), where p_i and q_i are indices of the points of A and B. Indices must be increasing and must include each index exactly once for both A and B. Value $\perp$ means that a particular point was not matched. Example of a valid alignment for $n^A = 4, n^B = 5$:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & \perp & \perp \\ 1 & \perp & 2 & 3 & 4 & 5 \end{bmatrix}$$

$D(A, B, M) = \sum\limits_{(p,q) \in M} d((r_p^A, w_p^A), (r_q^B, w_q^B))$

The best alignment $M^*(A, B)$ is the alignment M which minimizes $D(A, B, M)$.

The distance of A and B is $D^*(A, B) = D(A, B, M^*(A, B))$.

The best alignment is found by dynamic programming (this is basically the Needleman-Wunsch algorithm [cite]) – for this, the distance function d is converted into a score function s:

$$s((r_i^A, w_i^A), (r_j^B, w_j^B)) = \frac{1}{2}w_i^A + \frac{1}{2}w_j^B - d((r_i^A, w_i^A), (r_j^B, w_j^B))$$

$$s((r_i^A, w_i^A), \perp) = 0 \qquad s(\perp, (r_j^B, w_j^B)) = 0$$

$$S(A, B, M) = \sum_{(p,q) \in M} s((r_p^A, w_p^A), (r_q^B, w_q^B))$$

$$S(A, B, M) = \sum_{i=1}^{n^A} w_i^A + \sum_{j=1}^{n^B} w_j^B - D(A, B, M)$$

From this equation it can be seen, that maximizing S by dynamic programming also minimizes D.

A useful property of the distance function D^* is that it is a metric (i.e. $D^*(A, A) = 0$, $D^*(A, B) = D^*(B, A)$, and $D^*(A, B) + D^*(B, C) \geq D^*(A, C)$ for any weighted structures A, B, C).

Having two structures A, B and their best alignment $M^*(A, B) = [(p_1, q_1), ..., (p_n, q_n)]$, we can define operation $merge$ as follows:

$$merge(A, B) = C = (n^C, R^C, W^C, k^C)$$

$$n^C = n$$

$$r_i^C = \frac{r_{p_i}^A w_{p_i}^A k^A + r_{q_i}^B w_{q_i}^B k^B}{w_{p_i}^A k^A + w_{q_i}^B k^B}$$

$$w_i^C = w_{p_i}^A k^A + w_{q_i}^B k^B$$

$$k^C = k^A + k^B$$

(If $p_i = \perp$, the values can be calculated by setting $w_{p_i}^A = 0$, thus simplifying to $r_i^C = r_{q_i}^B, w_i^C = w_{q_i}^B$. Similarly for $q_i = \perp$.)

Remark: When finding the two nearest items in the work set, it is not necessary to calculate the distance D^* for every pair of items – there are specialized data structures that can significantly decrease the number of distance calculations. We use a non-standard structure NN-tree (nearest neighbour tree). (Standard structures like VP-tree, GH-tree, GNAT, M-tree etc. (TODO name, what I tried) either miss some of the necessary operations (insert, delete) or perform worse than NN-tree for this particular application.) In some larger protein families this can reduce the number of distance computations to roughly 20%.

7 References

Exemplini F *et al.* (2019) Examples in scientific literature. J Exemp Biol, 42, 1-100.