OverProt – Description of methods

Table of contents

1 Introduction

OverProt is a tool for constructing and visualizing the secondary structure consensus for protein families. The consensus produced by OverProt can be used as a template for annotation of secondary structure elements in protein families, e.g. 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.

The source code is freely available at `https://gitlab.com/midlik/overprot`.

2 Terminology

- **Protein structure** – a set of atoms with assigned 3D coordinates. A structure consists of one or more **chains**. A chain is a sequence of **residues**, each of which consists of the individual **atoms**. OverProt works with structures in **mmCIF format**. Structures deposited in the PDB (Armstrong *et al.*, 2020) 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 in some cases different from the *auth** numbering scheme.
- **Protein domain** – a part of protein structure, either a whole chain or a range (ranges) of residues in a chain. A domain is defined by the structure identifier (PDB ID), 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 a reasonable structural similarity. The set can be provided by the user or it can be defined based on the CATH database (Sillitoe *et al.*, 2021), 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 key 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 (H/E). 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 (realized by hydrogen bonds) and can be either parallel or antiparallel; a **β-sheet** is a set of strands which are connected by β-ladders (a connected component).
  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 (detection of SSEs) is performed by **SecStrAnnotator**, more details can be found in its original paper (Midlik *et al.*, 2019).

We will sometimes use the term **base SSEs** to distinguish SSEs from consensus SSEs.

- **Consensus SSE** – a set of equivalent SSEs from different family members.
- **Secondary structure consensus** – a set of consensus SSEs with a given 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 several steps. (In the following text, `--xx` refers to a command-line option of `overprot.py`, `[xx]yy` refers to a setting yy in section xx in the configuration file (`overprot-config.ini`).)

3.1 Preparation

- The list of domains for the family is downloaded from PDBe API `https://www.ebi.ac.uk/pdbe/api/mappings/{family_id}` (if not already given by `--domains`).
- Select sample: If `--sample_size` is smaller that the number of domains, a random subset of the domain list is selected.
  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; the 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`. The structures are also converted to the PDB format for later steps (namely, for alignment by program MAPSCI). The download step is performed by an auxiliary program `StructureCutter` written in C# (a part of the OverProt project).

3.2 Structural alignment

Multiple structure alignment is performed in 2 steps:

- Program MAPSCI (Ilinkin *et al.*, 2020) 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 it selects the same subset every time).
  To reduce indeterminism and ease later visualization, the consensus structure is centered to the origin (0, 0, 0), rotated so that its PCA (principal component analysis) components are aligned to the XYZ axes (“the structure is laid flat”), and flipped in a consistent way (roughly so that the start and the end of the chain are more in front, and the chain goes from left-top to right-bottom).

In general, MAPSCI produces a reasonable consensus structure, but the alignment of the individual domains is often poor, so the following re-alignment step is necessary.

- In the re-alignment step, all domains are structurally aligned onto the MAPSCI consensus structure via `cealign` algorithm (Shindyalov and Bourne, 1998) provided in PyMOL module (Schrödinger, LLC.) version 2.3.0. In rare cases `cealign` fails (when the domains are too short) – in such cases a simple internal algorithm is used instead (theoretically inefficient and not allowing gaps, but sufficient for these very short domains).

3.3 Secondary structure assignment

The SSEs in each domain are detected by `SecStrAnnotator` (Midlik *et al.*, 2019, 2021) with options `--onlyssa --verbose --batch`.

3.4 Guide tree

The domains are clustered by agglomerative clustering to produce a **guide tree**. The algorithm starts with a set of structures. It finds the two most similar structures and merges them into a new structure. This is then repeated until we end up with a single structure corresponding to the tree root.

This agglomerative algorithm can be expressed by the following pseudocode:

```
Workset = { the structures of all input domains }
while |Workset| > 1:
    A, B = two nearest structures in Workset
    C = merge_structures(A, B)
    Children of C = {A, B}
    Workset = Workset - {A, B} ∪ {C}
```

At the end, `Workset` will only contain one structure, which is the tree root. The topology of the tree will be defined by `Children`. An example is shown in Figure 1.

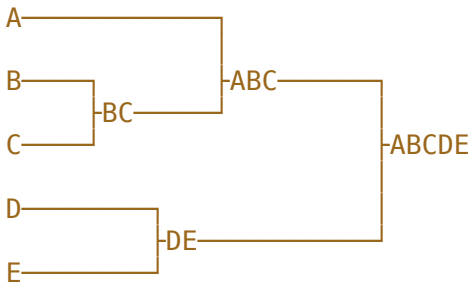

Figure 1: An example of the guide tree construction. 5 structures were initially in `Workset`. B+C were merged into BC, then D+E into DE, then A+BC into ABC, and finally ABC+DE into ABCDE.

The details of the algorithm are described in Appendix 6.1 (distance function, which determines the nearest structures) and 6.2 (operation `merge_structures`).

3.5 Merging

This step is the core of the consensus generation algorithm. As an 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 (see Figure 2).

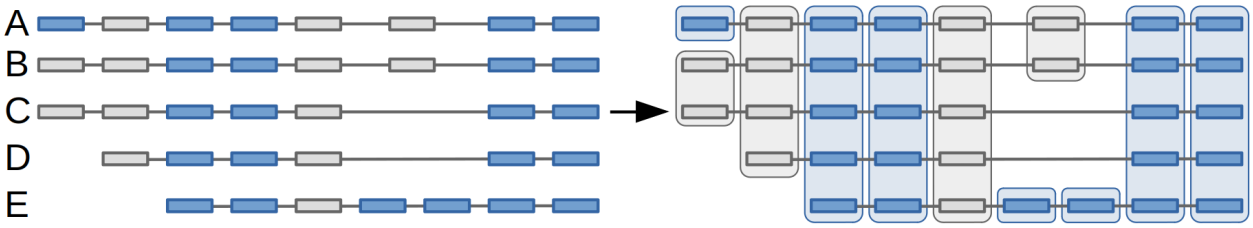

Figure 2: An example of 5 domains simplified to a sequence of base SSEs (gray = helix, blue = strand) and their clustering into 11 clusters.

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 order of the clusters. This constraint can be formalized as:
   - Base SSE x precedes base SSE y (written $x \to y$) if they are from the same protein domain and x goes before y in the sequence.
   - Cluster P directly precedes cluster Q ($P \Rightarrow Q$) if there exist SSEs $x \in P, y \in Q$ such that $x \to y$.
   - Cluster P precedes cluster Q ($P \to Q$) if there exists a sequence of clusters $P \Rightarrow R_1 \Rightarrow ... \Rightarrow R_n = Q$ where $n \geq 1$ (in other words, $\to$ is the transitive closure of $\Rightarrow$).
   - There must be no cluster P, such that $P \to P$.

Note: The order of some clusters may be undefined (i.e. neither $P \to Q$ nor $Q \to P$) if they contain no SSEs from the same domain. Therefore $\to$ is a partial order on the clusters (not a total order). We represent the order by a directed acyclic graph (DAG) (see Figure 3).

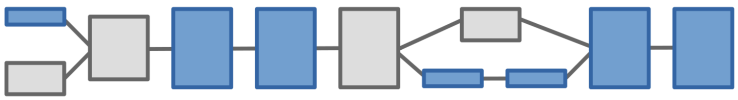

Figure 3: An example of a DAG representing clusters of SSEs from Figure 2. The height of each rectangle shows the weight of the cluster (the number of base SSEs in the cluster). The color shows the cluster type (gray = helix, blue = strand). The direction of the edges is implicit (left to right). The egdes that can be inferred from transitivity are not shown (i.e. we show only the transitive reduction (Hasse diagram)).

The merging step follows the guide tree. First, each guide tree leaf is populated with the DAG of SSEs of the respective domain. In each internal node, the DAGs from the two children nodes are matched together and merged. The root then contains the consensus SSEs of the whole family (see Figure 4).

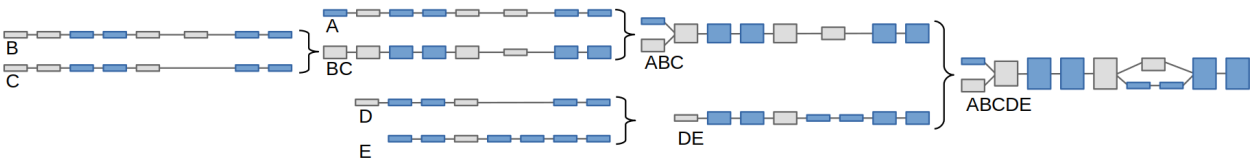

Figure 4: The process of merging 5 DAGs based on the guide tree from Figure 1.

The matching and merging of two SSE DAGs is in principle similar to matching and merging of two weighted structures. The best matching is also found by dynamic programming. However, it is more complicated here because 1) SSEs of different type cannot be matched (this can cause the branching in the resulting DAG), and 2) the dynamic programming algorithm is not as straightforward for matching DAGs as it is for matching sequences. More details are provided in Appendix 6.3.

The β-connectivity is not directly considered in the merging algorithm (though it is included in the distance function for DAG matching). Therefore it is necessary to determine the β-connectivity of the resulting clusters based on the β-connectivity of the base SSEs.

A β-ladder PQo (connecting strand clusters P and Q with orientation o (parallel/antiparallel)) is included in the resulting consensus if

$$\frac{n_{PQo}}{\min\{n_P, n_Q\}} \geq 0.5$$

where n_P is the number of strands in cluster P, n_Q is the number of strands in cluster Q, and n_{PQo} is the number of base ladders connecting a base strand in P to a base strand in Q with orientation o (see Figure 5).

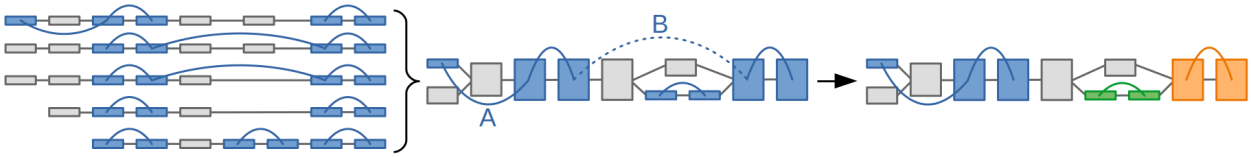

Figure 5: Merging β-ladders from 5 domains. Lower arcs show parallel, upper arcs antiparallel ladders. Ladder A is included because $n_{PQo}/\min\{n_P, n_Q\} = 1/\min\{1, 5\} = 1 \geq 0.5$. Ladder B is not included because $n_{PQo}/\min\{n_P, n_Q\} = 2/\min\{5, 5\} = 0.4 < 0.5$. The rightmost column shows the separation of the consensus strands into sheets (connected components).

After the clustering, a variety of statistics are computed for each consensus SSE and saved in `results/consensus.sses.json`:

- Occurrence – the number of domains that contain this SSE, divided by the total number of domains in the family. In the previous example, the first strand occurs in 1 out of 5 domains; thus its occurrence is 0.2 or 20%.
- Average length – measured as the number of residues.
- Average line segment – the average start and end point in 3D.
- 3D variability – the variance of the start and end point.

Each consensus SSE also gets a unique label (containing its type and sequential number, e.g. E0, H1, H2...) and color.

3.6 Annotation

In this optional step, the generated SSE consensus is used as an 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. SecStrAnnotator is run with these options: `--ssa file --align none --metrictype 3 --fallback 30 --unannotated`. Metric type 3 must be used because the default metric requires residue numbers for each SSE, but these are not available for the consensus SSEs. Option `--unannotated` includes also the unannotated SSEs in the resulting annotation files, with labels prefixed by underscore (e.g. `_H0`).

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 by OverProt Viewer. A PyMOL session (.pse) is created, with the MAPSCI consensus structure shown as ribbon and the consensus SSEs shown as cylinders and arrows. The width of each cylinder/arrow shows the occurrence of the corresponding helix/strand. 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 Execution

OverProt Core is implemented mostly in Python3 and designed to run in the Linux environment (tested on Ubuntu 20.04). On the other operating systems, it can be run in Docker. Before the first execution, the dependencies must be installed:

```
sh install.sh --clean
```

All steps of the algorithm are combined in `overprot.py`. It is run in a Python virtual environment. Its arguments are the CATH family ID and the output directory:

```
. venv/bin/activate
python overprot.py --help
python overprot.py 1.10.630.10 data/cyp/
```

Multiple families can be processed in parallel using `overprot_multifamily.py`. Its arguments are the family list and the output directory:

```
. venv/bin/activate
python overprot_multifamily.py --help
python overprot_multifamily.py data/families.txt data/multifamily/
```

More details can be found in the `README.md` files in the project repository.

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.

OverProt Viewer shows each consensus SSE as a rectangle or an oval, whose height corresponds to its occurrence and width corresponds to its average length (number of residues). Strands from the same β-sheet are shown in the same color; helices are shown in gray. Connections of the strands in a β-sheet are shown by arcs (lower arcs – parallel, upper arcs – antiparallel). Hovering over an SSE shape shows the SSE details (see Figure 6).

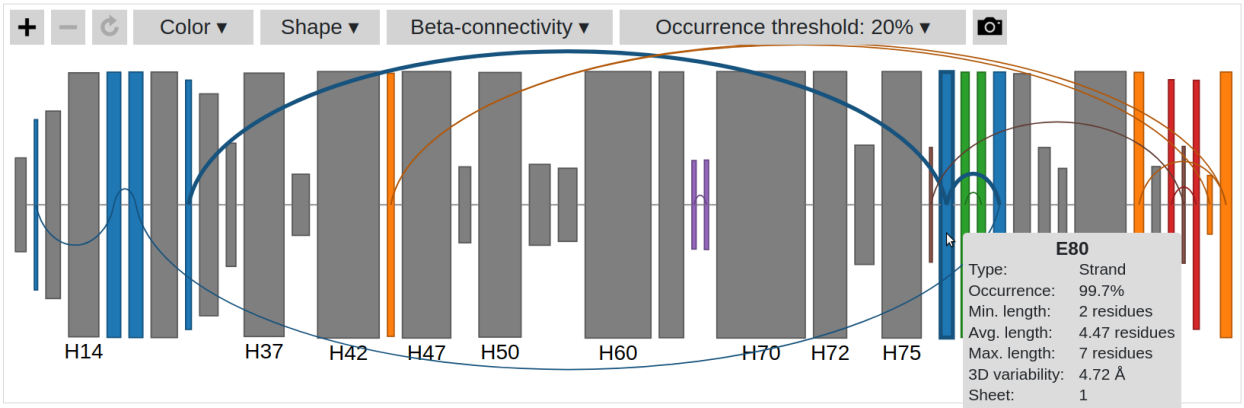

Figure 6: OverProt Viewer showing the secondary structure consensus for CATH family 1.10.630.10 (Cytochrome P450).

Visualization options include:

- Color:
  - Uniform – Show all SSEs in the same color.
  - Type – Show β-strands in blue, helices in gray.
  - Sheet – Assign the same color to all β-strands from the same β-sheet; show helices in gray.
  - Variability – The 3D variability measures the standard deviation of the SSE start and end point coordinates. Low values (dark) indicate conserved SSE position, high values (bright) indicate variable SSE position.
  - Rainbow – Standard rainbow coloring from N-terminus (blue) to C-terminus (red).
- Shape:
  - Rectangle – Show the SSEs as rectangles. The height of the rectangle indicates its occurrence; the width indicates its average length (number of residues).
  - SymCDF – The cumulative distribution function (CDF) describes the statistical distribution of the SSE length. The SymCDF shape consists of four symmetrical copies of the CDF; the bottom right quarter is the classical CDF. The widest part of the shape corresponds to the maximum length, the narrowest to the minimum length, the height corresponds to the occurrence.

- Beta-connectivity:
  - On – The beta-connectivity shows how β-strands are connected to each other in β-sheets. The lower arcs indicate parallel ladders; the upper arcs indicate antiparallel ladders.
  - Off – The beta-connectivity arcs are hidden.
- Occurrence threshold:
  - Hides the SSEs with occurrence lower than the specified threshold. Can be set to any number from 0% to 100%.

OverProt Viewer can be set to dispatch and listen to HTML events. When an SSE is hovered over or clicked, the viewer dispatches an event (`PDB.overprot.hover` or `PDB.overprot.select`). The information about the selected elements is included in `event.detail`. Conversely, the viewer handles the incoming events (`PDB.overprot.do.hover` or `PDB.overprot.do.select`) by highlighting the selected elements. This allows interactivity across several web components, as demonstrated by the integrated view in the OverProt web (`https://overprot.ncbr.muni.cz/domain_view?family_id=1.10.630.10&domain_id=1jfbA00`) – hovering over an SSE in any of the three components (OverProt Viewer, interactive 2DProts (Hutařová *et al.*, 2021), MolStar Viewer (Sehnal *et al.*, 2021)) highlights it in all three.

5 Data computation for OverProt Server

OverProt Server provides precomputed SSE consensuses (database) and runs the OverProt Core algorithm for user-defined sets of domains (jobs). OverProt Server is implemented using Python (Flask), Gunicorn, Redis Queue, Nginx, and Docker. A running instance is available at `https://overprot.ncbr.muni.cz`.

The database is constructed in this way:

- Retrieve the current list of families from CATH (`http://download.cathdb.info/cath/releases/latest-release/cath-classification-data/cath-superfamily-list.txt`). The list currently contains 6631 families, out of which 64 are empty families (January 2022).
- 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 in total (January 2022).
- Remove duplicates (i.e. multiple domains from the same PDB entry). The number of domains without duplicates is currently over 200k (January 2022).
- Apply the OverProt Core algorithm to each family.

The whole process is realized by:

```
. venv/bin/activate
python  overprot_multifamily.py  --download_family_list_by_size \
```

```
--config working_scripts/overprot-config-overprotserverdb.ini \
--collect  -  $UPDATE_DIRECTORY
```

6 Appendix

6.1 Distance function for two weighted structures

To be able to compare the real structures of the input domains as well as the artificial structures created by merging, we use the concept of a weighted structure. A weighted structure is a sequence of points (C-alpha coordinates) where each point has its relative weight. In any real structure, all relative weights are equal to 1, but merging can create points with smaller relative weights. The absolute weight of a weighted structure is simply the number of the real structures that have been merged to form this weighted structure.

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

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

$\mathbf{r}_i^A$ and w_i^A will refer to i-th column of $\mathbf{R}^A$ and $\mathbf{W}^A$.

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

The distance function d is defined for two weighted points:

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

The parameter R_0 was set to 10 Å.

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

$$d\left((\mathbf{r}_i^A, w_i^A), \perp\right) = \frac{1}{2} w_i^A \qquad d\left(\perp, (\mathbf{r}_j^B, w_j^B)\right) = \frac{1}{2} w_j^B$$

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

A matching (or alignment) of two weighted structures A, B is a sequence of pairs $[(p_1, q_1), (p_2, q_2), \ldots (p_n, q_n)]$, 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 matching for $n^A = 4, n^B = 5$:

$$[(1, 1),\ (2, \perp),\ (3, 2),\ (4, 3),\ (\perp, 4),\ (\perp, 5)]$$

The distance function D for two weighted structures A and B with a given matching M is defined:

$$D(A,B,M) = \sum_{(p,q)\in M} d\left((\mathbf{r}_p^A, w_p^A), (\mathbf{r}_q^B, w_q^B)\right)$$

The distance function D^* of two weighted structures A and B is then:

$$D^*(A,B) = D(A,B,M^*)$$

where M^* is the best matching of A and B, i.e. the matching which minimizes $D(A,B,M^*)$.

The best matching can be found by dynamic programming. For this, the distance function d is converted into the score function s:

$$s\left((\mathbf{r}_i^A, w_i^A), (\mathbf{r}_j^B, w_j^B)\right) = \frac{1}{2}w_i^A + \frac{1}{2}w_j^B - d\left((\mathbf{r}_i^A, w_i^A), (\mathbf{r}_j^B, w_j^B)\right)$$

$$s\left((\mathbf{r}_i^A, w_i^A), \perp\right) = 0 \qquad s\left(\perp, (\mathbf{r}_j^B, w_j^B)\right) = 0$$

Similarly, D is converted into the total score function S:

$$S(A,B,M) = \sum_{(p,q)\in M} s\left((\mathbf{r}_p^A, w_p^A), (\mathbf{r}_q^B, w_q^B)\right) = \frac{1}{2}\sum_{i=1}^{n^A} w_i^A + \frac{1}{2}\sum_{j=1}^{n^B} w_j^B - D(A,B,M)$$

From this equation, it can be seen that maximizing S by dynamic programming also minimizes D. (This dynamic programming algorithm is in principle very similar to the well-known Needleman-Wunsch algorithm for aligning sequences (Needleman and Wunsch, 1970), but it differs in the score function it uses.)

Notes: The distance function D^* is inspired by the edit distance for comparing two strings. It basically measures how much we have to edit A (move/insert/delete points) to transform it into B. Thanks to this design, D^* 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).

When finding the two nearest items in the workset, 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 neighbor tree). In some larger protein families, this can reduce the number of distance computations to less than 20%. (Standard structures like GH-tree, M-tree, etc. either miss some of the necessary operations (insert, delete) or perform worse than NN-tree for this particular application.) This is only possible because D^* is a metric.

6.2 Merging two weighted structures

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

$$merge\_structures(A, B) = C = (n^C, \mathbf{R}^C, \mathbf{W}^C, k^C)$$

$$n^C = n$$

$$\mathbf{r}_i^C = \frac{\mathbf{r}_{p_i}^A w_{p_i}^A k^A + \mathbf{r}_{q_i}^B w_{q_i}^B k^B}{w_{p_i}^A k^A + w_{q_i}^B k^B}$$

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

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

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

6.3 Matching two SSE directed acyclic graphs (DAGs)

The distance function d for two SSEs P and Q is defined as the sum of Euclidean distances between their start points and between their end points:

$$d(P, Q) = \|\mathbf{u}_P - \mathbf{u}_Q\| + \|\mathbf{v}_P - \mathbf{v}_Q\|$$

where $\mathbf{u}_P, \mathbf{v}_P$ is the start and end point of SSE P, $\mathbf{u}_Q, \mathbf{v}_Q$ is the start and end point of SSE Q.

The score function s is then defined:

$$s(P, Q) = \begin{cases} SR(d(P, Q)) & \text{if } P, Q \text{ are of the same type (helix/strand)} \\ 0 & \text{otherwise} \end{cases}$$

where SR is the "smoothed ramp" function, which is basically a smooth, strictly decreasing version of the function $y = \max\{0, 1 - x/d_0\}$.
SR is defined by the implicit equation $d_0(1-\alpha)y^2 + (x + d_0(2\alpha - 1))y - d_0\alpha = 0$. When solving this quadratic equation, the greater root is selected. The parameters were set to $d_0 = 30$ Å and $\alpha = 0.01$.

The distance function d and the score function s can be easily extended from base SSEs to consensus SSEs. For a consensus SSE P, the point $\mathbf{u}_P$ is simply the arithmetic mean of $\mathbf{u}$ of all base SSEs included in P. Similarly for $\mathbf{v}_P$.

However, it will be useful to define the weight of a consensus SSE P (w_P) as the number of base SSEs included in P. Similarly, we will define the weights of the consensus β-ladders: w_{PQp} is the number of parallel ladders connecting a base strand in P to a base strand in Q, w_{PQa} is the number of antiparallel ladders connecting a base strand in P to a base strand in Q. (Base SSEs/ladders can be understood as consensus SSEs/ladders with weight 1.)

In order to reflect the β-connectivity in the score function, the "ladder correction" is applied to the strands:

$$s_{\text{corr}}(P_i, Q_j) = \frac{1}{2}\left(s(P_i, Q_j) + \sum_k \sum_l (\alpha_{ijkl} + \beta_{ijkl}) s(P_k, Q_l)\right)$$

where P_i, P_k are strands in the first matched DAG, Q_j, Q_l are strands in the second matched DAG, and the coefficients $\alpha_{ijkl}, \beta_{ijkl}$ maximize the value of $s_{\text{corr}}(P_i, Q_j)$ while fulfilling the following constraints:

$$\alpha_{ijkl} \geq 0 \qquad \beta_{ijkl} \geq 0$$

$$\sum_k \sum_l (\alpha_{ijkl} + \beta_{ijkl}) \leq 1$$

$$\sum_l \alpha_{ijkl} \leq \frac{w_{P_i P_k a}}{w_{P_i}} \qquad \sum_l \beta ijkl \leq \frac{w_{P_i P_k p}}{w_{P_i}}$$

$$\sum_k \alpha_{ijkl} \leq \frac{w_{Q_j Q_l a}}{w_{Q_j}} \qquad \sum_k \beta ijkl \leq \frac{w_{Q_j Q_l p}}{w_{Q_j}}$$

For each pair P_i, Q_j, the values of coefficients $\alpha_{ijkl}, \beta_{ijkl}$ are determined by a greedy algorithm (i.e. first assigning the greatest possible value to the coefficients corresponding to the highest $s(P_k, Q_l)$, then the second highest, etc.).

For helices, no "ladder correction" is necessary, so $s_{\text{corr}}(P_i, Q_j) = s(P_i, Q_j)$.

A matching of two SSE DAGs $\boldsymbol{G}, \boldsymbol{H}$ is a set of pairs $M = \{(P_1, Q_1), (P_2, Q_2), ...(P_n, Q_n)\}$, where $P_i \in V(\boldsymbol{G}), Q_j \in V(\boldsymbol{H})$, fulfilling these conditions:

- Each vertex is matched at most once: $\forall i, j : P_i \neq P_j \Leftrightarrow Q_i \neq Q_j$
- Only vertices of the same type are matched: $\forall i : \text{type}(P_i) = \text{type}(Q_i)$
- No cycle is created: $\nexists i, j : P_i \to P_j \wedge Q_j \to Q_i$

The best matching M^* of DAGs $\boldsymbol{G}, \boldsymbol{H}$ is the matching which maximizes the total score S:

$$S(\boldsymbol{G}, \boldsymbol{H}, M^*) = \sum_{(P,Q) \in M^*} w_P w_Q s_{\text{corr}}(P, Q)$$

The corresponding best score is S^*:

$$S^*(\boldsymbol{G}, \boldsymbol{H}) = S(\boldsymbol{G}, \boldsymbol{H}, M^*)$$

The problem of finding the best matching and the best score for two DAGs $\boldsymbol{G}, \boldsymbol{H}$ can be decomposed to smaller problems:

$$\begin{aligned} S^*(\boldsymbol{G}, \boldsymbol{H}) = \max \Big(& \{S^*(\boldsymbol{G} - P, \boldsymbol{H}) \mid P \in \text{sinks}(\boldsymbol{G})\} \\ & \cup \{S^*(\boldsymbol{G}, \boldsymbol{H} - Q) \mid Q \in \text{sinks}(\boldsymbol{H})\} \\ & \cup \{S^*(\boldsymbol{G} - P, \boldsymbol{H} - Q) + w_P w_Q s_{\text{corr}}(P, Q) \mid P \in \text{sinks}(\boldsymbol{G}), Q \in \text{sinks}(\boldsymbol{H})\} \Big) \end{aligned}$$

The trivial subproblems can be solved directly without decomposition:

$$S^*(\boldsymbol{G}, \boldsymbol{K_0}) = S^*(\boldsymbol{K_0}, \boldsymbol{H}) = 0 \qquad\qquad M^*(\boldsymbol{G}, \boldsymbol{K_0}) = M^*(\boldsymbol{K_0}, \boldsymbol{H}) = \{\}$$

where $\boldsymbol{K_0}$ is a graph with no vertices.

OverProt finds the best matching by a dynamic programming algorithm based on the described decomposition.

After the best matching is found, the matched pairs of vertices are merged. The resulting DAG contains the merged matched vertices plus the nonmatched vertices from the original DAGs $\boldsymbol{G}, \boldsymbol{H}$. The edges are merged accordingly, and transitive closure is applied. If vertices P, Q are matched and merged into a vertex R, then:

$$w_R = w_P + w_Q \qquad \mathbf{u}_R = \frac{\mathbf{u}_P w_P + \mathbf{u}_Q w_Q}{w_P + w_Q} \qquad \mathbf{v}_R = \frac{\mathbf{v}_P w_P + \mathbf{v}_Q w_Q}{w_P + w_Q}$$

7 References

Armstrong,D.R. *et al.* (2020) PDBe: improved findability of macromolecular structure data in the PDB. *Nucleic Acids Res*, **48**, D335–D343. https://doi.org/10.1093/nar/gkz990

Hutařová Vařeková,I. *et al.* (2021) 2DProts: database of family-wide protein secondary structure diagrams. *Bioinformatics*, **37**, 4599–4601. https://doi.org/10.1093/bioinformatics/btab505

Ilinkin,I. *et al.* (2010) Multiple structure alignment and consensus identification for proteins. *BMC Bioinformatics*, **11**, 71. https://doi.org/10.1186/1471-2105-11-71

Midlik,A. *et al.* (2019) Automated family-wide annotation of secondary structure elements. *Methods Mol Biol*, **1958**, 47–71. https://doi.org/10.1007/978-1-4939-9161-7_3

Midlik,A. *et al.* (2021) Uncovering of cytochrome P450 anatomy by SecStrAnnotator. *Sci Rep*, **11**, 12345. https://doi.org/10.1038/s41598-021-91494-8

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, **48**, 443–453. https://doi.org/10.1016/0022-2836(70)90057-4

Schrödinger, LLC. The PyMOL Molecular Graphics System, Version 2.3 https://pymol.org/

Sehnal,D. *et al.* (2021) Mol* Viewer: modern web app for 3D visualization and analysis of large biomolecular structures. *Nucleic Acids Res*, **49**, W431–W437. https://doi.org/10.1093/nar/gkab314

Shindyalov,I.N. and Bourne,P.E. (1998) Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng*, **11**, 739–747. https://doi.org/10.1093/protein/11.9.739

Sillitoe,I. *et al.* (2021) CATH: increased structural coverage of functional space. *Nucleic Acids Res*, **49**, D266–D273. https://doi.org/10.1093/nar/gkaa1079