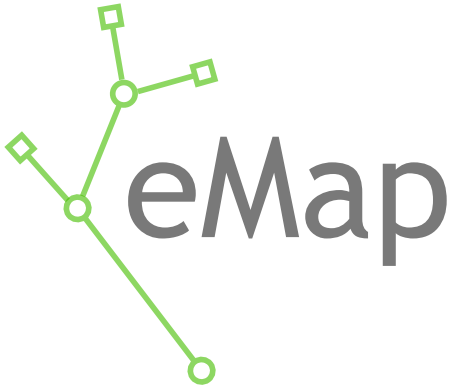

eMap
Version 2.0
User Manual

Contents

1 Introduction

eMap is a Python-based web application enabling automatic identification and visualization of possible electron or hole transfer channels in proteins based on their crystal structure using graph theory. eMap 2.0 allows for two modes of the analysis: (i) identifying electron transfer pathways in a single protein; and (ii) searching for common electron transfer pathways in a series of proteins. The back-end is powered by the open-source python package PyeMap.

Single protein analysis is based on a coarse-grained version of Beratan and Onuchic's Pathway model[1,2] and only accounts for the through-space *hopping* between aromatic residues side chains. Side chains of aromatic residues and non-protein electron transfer active moieties are modeled as vertices in a weighted graph, where the edge weights are modified distance-dependent penalty functions. Shortest path algorithms are used to compute the shortest pathways from a specified electron or hole donor to the surface of the protein, or to a user-specified acceptor. Predicted pathways can be visualized within the two-dimensional aromatic residue network and in 3D by means of NGL viewer.[3]

Multi-protein screening for common electron pathways exploits subgraph mining techniques using the protein graphs generated identically to the single protein case. Protein graph mining is based on the gSpan algorithm[4] and its Python implementation (https://github.com/betterenvi/gSpan). Shared graph patterns are further mapped onto specific subgraphs in individual proteins and clustered based on their sequence or structural similarity. Similar to the single protein case, the user can visualize the identified shared electron transfer pathways in 2D as a graph or in 3D by means of NGL viewer.[3]

This manual provides a basic description of the user interface and capabilities of the software, and outlines the algorithms involved at different stages of the analysis. For more information on PyeMap, which also serves as a standalone tool, please see the documentation.

To cite please use the following reference:

R.N. Tazhigulov, J.R. Gayvert, M. Wei, and K.B. Bravaya, eMap: A Web Application for Identifying and Visualizing Electron or Hole Hopping Pathways in Proteins. *J Phys. Chem. B* **2019**, 123, 32, 6946-6951. DOI: 10.2021/acs.jpcb.9b04816

Any questions or feedback can be sent to `emap.bu@gmail.com`.

2 Single protein analysis

2.1 Usage

This section describes the functionality and the user interface of the single protein analysis tool. The analysis includes three main steps: (i) parsing a user-specified .pdb or .cif file and identifying all moieties potentially involved in electron or hole transfer (aromatic side chains, aromatic fragments of co-factors, metals and metal clusters, user-specified molecular fragments); (ii) constructing the graph theory model of the protein crystal structure; and (iii) searching for the shortest paths connecting a user-defined electron or hole donor to a protein surface-exposed residues or a user-defined terminal electron or hole acceptor. The user interface and various options associated with each of the steps are described in Sections 2.1.1, 2.1.2, and 2.1.3, respectively.

2.1.1 File Upload and Parsing

Users may upload their own protein structures or fetch them by their unique 4-character PDB ID from the RCSB protein data bank (Fig. 1). Uploaded structures must adhere to either the Crystallographic Information File (CIF) or Protein Data Bank (PDB) file formats to be analyzed. In addition to standard protein residues, eMap will automatically identify non-protein electron/hole transfer active (ET active) moieties. Once parsing is complete, the user can select which protein chains, aromatic protein residues, and automatically identified non-protein ET active moieties to include in the analysis. See Section 2.3.2 for more details.

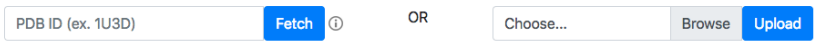

Figure 1: Panel for fetching/uploading PDB file.

2.1.2 Graph Construction

The second step is the specification of various options that determine how the graph theory model of the protein structure is constructed (Fig. 2). Each edge connecting two vertices is assigned a weight $P' = -\log(\epsilon)$, where ϵ is a distance-dependent hopping penalty function (see Sec 2.3.4). After processing, the generated 2D graph image will appear on the webpage. In the graph image, surface-exposed residues are indicated as rectangular nodes, while buried residues appear as oval nodes. Below, the parameters defining a way the graph is constructed are listed.

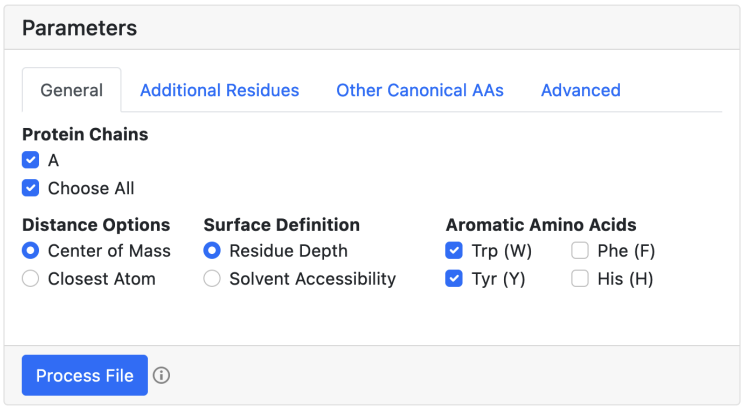

Figure 2: Panel for selection of parameters to construct the graph theory model.

General

Protein Chains: Choose which protein chains to include in the analysis. Default: all chains.

Distance Options: Distance between two ET active moieties can be calculated as the distance between the centers of mass, or as the distance between the closest atoms. Default: centers of mass.

Surface Definition: Residues can be classified as buried/exposed using one of two parameters: relative solvent accessibility (RSA)[1] or residue depth (RD). Default: RD. See Section 2.3.3 for more details.

Aromatic Amino Acids: Choose which standard amino acid residues to include in the analysis. Default: only tryptophans (W) and tyrosines (Y) are included.

Additional Residues

Non-Protein ET Active Moieties: Select which non-protein ET active moieties to include in the graph. Default: all automatically identified non-protein ET active moieties on the selected chains are included.

Custom Atom Range: Users may specify their own custom fragments atom-by-atom using eMap's custom fragment selection syntax, which is based on PDB atom serial number. See below for

[1] The current implementation of relative solvent accessibility (RSA) is unable to identify non-protein ET active moieties and custom fragments as surface-exposed, and therefore, these nodes will always be considered buried in the graph when using this algorithm.

more details.

Custom Atom Range

The custom fragment selection syntax is as follows:

Table 1: Custom Fragment Selection Syntax

","	defines discrete range of atoms
"-"	defines continuous range of atoms
"()"	encloses atom range for a single user-specified residue
"(...),(...)"	defines multiple custom fragments

Custom fragments are named "CUST-1", "CUST-2",... etc. based on the order they were specified. See Fig. 3 for an example.

Restrictions

Custom fragments may not contain atoms that are already included in the graph as part of a standard aromatic residue or automatically identified non-protein ET active moieties. Deselect any automatically detected non-protein ET active moieties if you wish to include one of its atoms as part of a user-specified custom fragment.

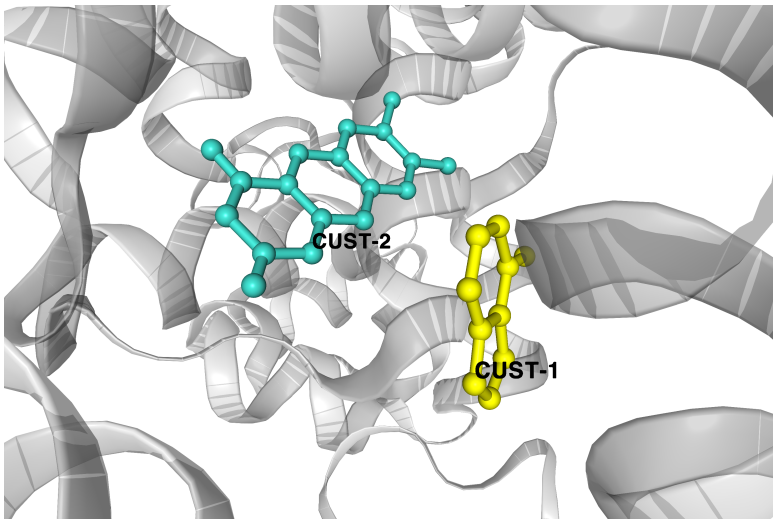

Figure 3: Two user-specified custom fragments in cryptochrome 1 from *A.thaliana* (PDB ID: 1U3D). The string '(3960-3969),(3970-3980,3982,3984-3987)' generates residues "CUST-1" and "CUST-2", respectively. Note that in this example, automatically detected non-protein ET active moieties FAD510-1 and FAD510-2 were deselected in order to allow selection of these atoms as part of user-specified custom fragments.

Other Canonical AAs

Choose which non-aromatic standard amino acid residues to include in the analysis. Default: None

Advanced

Distance Cutoff: Defines a pure distance threshold. eMap will only keep edges with distances less than or equal to the specified threshold. Default: 20Å

Edge Pruning Algorithm: Choice of algorithm for pruning edges from the graph. Default: Percent

Max Degree: Maximum number of edges per vertex. Only applies when *Edge Pruning Algorithm* is set to Degree. Default: 4

Edges per Vertex: Specifies a percentage of the shortest edges per vertex to keep (see Sec. 2.3.4). Only applies when *Edge Pruning Algorithm* is set to Percent. Default: top 1%

SD Cutoff: Specifies n, a vertex-specific distance threshold. Of the remaining edges not filtered out by the previous two settings, only those with length $l \leq \bar{l}_{vertex} + n\sigma_{vertex}$ are kept, where $\bar{l}_{vertex}$ is the mean of all edges associated with the vertex, σ_{vertex} is the standard deviation in length in the remaining set of edges for a given vertex. Only applies when *Edge Pruning Algorithm* is set to Percent. Default: $n = 1$

Residue Depth Cutoff: Threshold for classifying residues as buried or surface exposed when residue depth criteria is used. Default: 3.03 Å

RSA threshold: Threshold for classifying residues as buried or surface exposed when solvent accessibility criteria is used. Default: 0.05

Penalty Function Parameters: Modify the hopping parameters for the penalty functions $\epsilon = \alpha \exp\left[-\beta(R - R_{offset})\right]$. Default values ($\alpha = 1.0$, $\beta = 2.3$, $R_{offset} = 0.0$) result in a purely distance-dependent model.

Note: The *Edges per Vertex*, *Distance Cutoff*, and *SD Cutoff* thresholds are applied to raw distances rather than the edge weights, which are subsequently calculated based on the specified *Penalty Function Parameters*. See Section 2.3.4 for more details.

2.1.3 Identification of Shortest Pathways

The next step is specification of a source residue in order to calculate and visualize the shortest pathways to the surface (Fig. 4). Alternatively, the user may specify a source and target pair in order to calculate and visualize the shortest pathways from a source to a target. Each pathway identified by eMap is assigned a **Score**, which is simply the sum of the weights of the edges constituting the pathway.

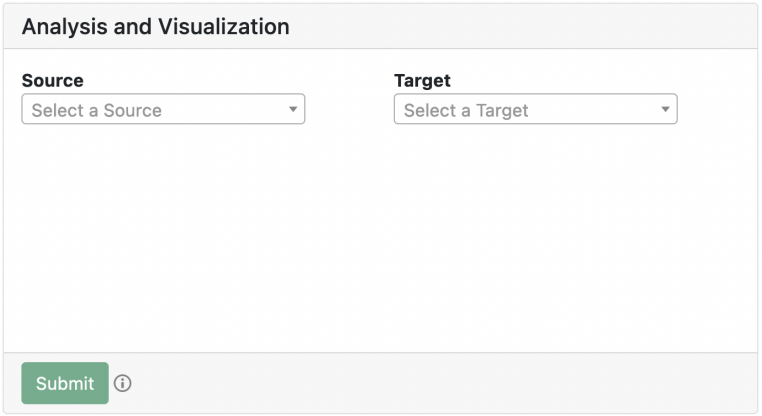

Figure 4: Panel for specifying a source and a target for computing shortest pathways.

Source only

When only a source node is selected, Dijkstra's algorithm is used to calculate the shortest path from the source to each surface-exposed residue. In the output, the pathways are organized into "branches" based on the first surface-exposed residue reached during the course of the pathway. Each pathway is assigned a unique pathway ID for visualization. The top 5 shortest pathways can be visualized in NGL viewer[3] using the automatically generated checkboxes. Information on all calculated pathways is presented in a table at the bottom of the webpage.

Specified Target

If a target is specified, a procedure based on Yen's Algorithm[5] is used to calculate the 5 shortest paths from source to target. The target does not need to be a surface-exposed residue. Each of these pathways is assigned a unique pathway ID for the purpose of visualization. Information on all calculated pathways is presented in a table at the bottom of the webpage.

2.2 Visualization of Pathways

Pathways are visualized in 2D as a graph, and in 3D as highlighted amino acid residues and ET active moieties in the protein structure (Figs. 6, 7) using NGL viewer.[3] Screenshots of the current NGL viewer frame and the 2D graph image are available for download.

After computing the shortest pathways, 6 checkboxes will appear above the NGL viewer (Fig. 5). The first is a cartoon representation of the entire protein. The remaining checkboxes correspond to the 5 shortest pathways computed by eMap. Users may choose to visualize any pathway not automatically visualized by selecting its pathway ID in the "Pathway ID" box. This pathway will be assigned a checkbox. Users are only able to select one non-automatically visualized pathway at a time. To see information on all available pathways, the user can click "View Pathways" to be taken to the table at the bottom of the webpage, which contains information on all calculated pathways. A data file containing all computed pathway information is also available for download.

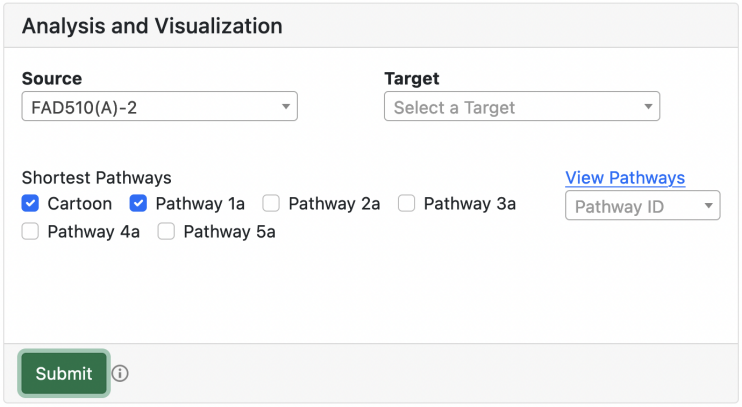

Figure 5: Panel for visualizing different pathways.

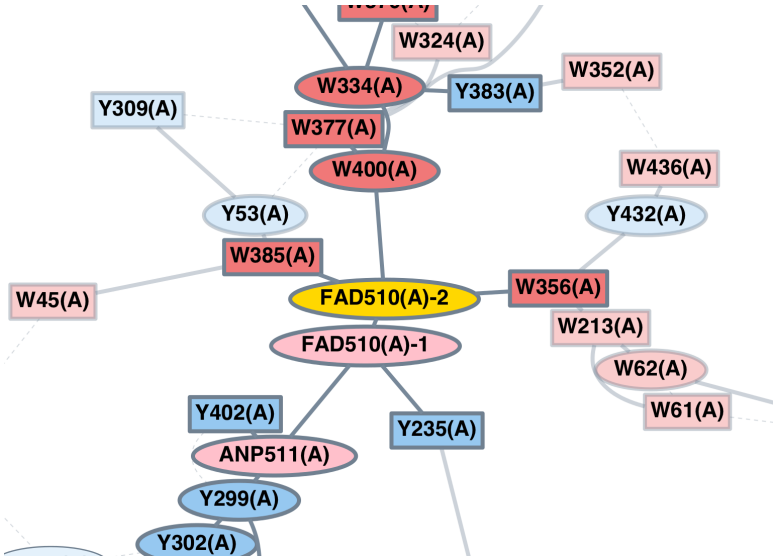

Figure 6: 2D graph image of pathways computed by eMap.

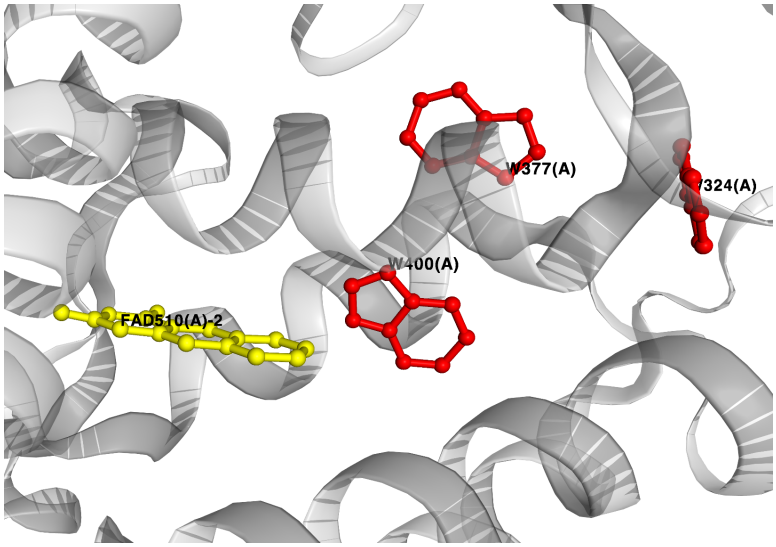

Figure 7: 3D image of cryptochrome 1 from *A.thaliana* (PDB ID: 1U3D) with ET Pathway "Flavin <–> W400 <–> W377 <–> W324" visualized in NGL Viewer.

2.3 Algorithms

This section outlines the algorithms involved at each stage of the analysis, and the relevant parameters used to construct the graph theory model.

2.3.1 Parsing the File

The analysis requires a valid .pdb or .cif, fetched from the PDB database or uploaded by the user, to be parsed by eMap. The PDB package of the open source Python library Biopython is employed to parse the fetched/uploaded file.[6]

2.3.2 Non-Protein ET Active Moieties

Frequently, protein crystal structures contain residues which are not amino acids, and do not belong to the polypeptide chain(s). Unless they are solvent molecules or salt ions not belonging to any co-factors, they can play significant role in electron/hole transfer. eMap automatically identifies those non-protein electron/hole transfer (ET) active moieties, and gives users the option to include them in the analysis.

In the current implementation, non-protein ET active moieties identified by eMap are non-amino acid aromatic sites or extended conjugated systems. For a given non-standard co-factor (e.g., flavin adenine dinucleotide), there can be multiple non-protein ET active moieties identified by eMap, and they will appear as separate nodes on the graph if selected for analysis.

Aromatic Moieties and Extended Conjugated Chains

After initial parsing, non-protein residues are analyzed for detection of ET active moieties. For each non-standard residue, a chemical graph is constructed using the NetworkX library, consisting of the O, C, N, P and S atoms in the residue.[7] To isolate the conjugated systems, an edge is only drawn between two atoms j and k if:

$$r_{\mathrm{jk}} \leq \ \overline{x} - 2\sigma_{\overline{x}}$$

where $\overline{x}$ is the mean single-bond distance between those two elements, and $\sigma_{\overline{x}}$ is the standard deviation. The data was obtained from the online CRC Handbook of Chemistry and Physics.[8]

If there are any conjugated systems, the resulting chemical graph will contain one or more connected components. Each connected component that contains a cycle, or consists of 10 or more atoms will be considered a non-protein ET active moiety, and can be selected for the analysis.

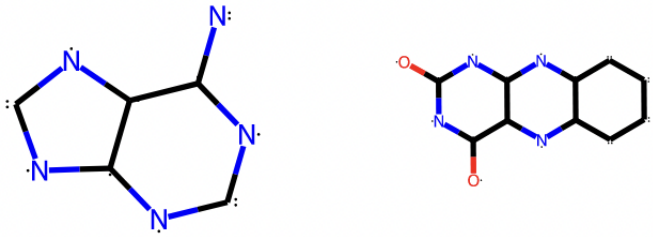

Figure 8: Adenine and flavin non-protein ET active moieties automatically identified in crystal structure (PDB: 1U3D) and visualized using RDKit.

Clusters

The PyeMap repository contains a list of 66 metal clusters which are automatically identified by their 3 character residue names. All atoms in the residue are collected, and a pre-rendered image is used for visualization of the chemical structure.

Metal ions

eMap automatically identifies a set of redox-active metal ions to include as residues in the graph.

Element	Charges
Cu	+1, +2, +3
Fe	+2, +3
Mn	+2, +3
Cr	+3
Ni	+2
Mo	0, +4, +6
Co	+2, +3

Visualization

In addition to identifying non-protein ET active moieties, eMap also provides an illustration of their chemical structures. This is done using the simplified molecular-input line-entry system (SMILES), and the open-source Cheminformatics tool RDKit.[9] We use the pysmiles package to generate the SMILES string for the chemical graph.

Prior to generating the SMILES string, eMap does post-processing of the chemical graph for non-protein ET-active residues in order to obtain proper chemical structures. Center of mass/closest atom distance calculations and 3D visualizations will be unaffected. For more details, please see the PyeMap documentation.

2.3.3 Identification of Surface-Exposed Residues

Electron (or hole) transfer often proceeds from/to surface residues to/from an acceptor/donor inside the protein. Therefore, identification of surface-exposed residues is a key step for prediction of relevant electron/hole transfer pathways. Users can select one of two parameters to classify residues as buried/exposed: residue depth and relative solvent accessibility.

Residue Depth

Residue depth is a measure of solvent exposure that describes the extent to which a residue is buried within the protein structure. The parameter was first introduced by Chakravarty and coworkers,[10] and is computed in eMap using the freely available program MSMS. MSMS computes a solvent-excluded surface by rolling a probe sphere along the surface of the protein, which is defined by atomic spheres. The boundary of the volume reachable by the probe is taken to be the solvent-excluded surface.[11] The residue depth for each residue is calculated as the average distance of its respective atoms from the solvent-excluded surface.

In eMap, the threshold for classifying residues as buried/exposed is:

$$\mathbf{RD} \leq \mathbf{3.03}$$

which is the threshold proposed by Tan and coworkers.[12] Residues 3.03 Å and shallower will be classified as exposed in the final graph; those deeper will be classified as buried. This threshold can be customized using the **Residue depth cutoff** slider under Advanced options.

For more information on MSMS, please see `http://mgltools.scripps.edu/packages/MSMS`.

Relative Solvent Accessibility

Accessible surface area is a measure of solvent exposure, first introduced by Lee and Richards,[13] which describes the surface area of a biomolecule that is accessible to solvent molecules. To calculate the accessible surface of each atom, a water sphere is rolled along the surface of the protein, making the maximum permitted van der Waals contacts without penetrating neighboring atoms.[14] The total accessible surface area for a residue is the sum of the solvent accessible surface areas of its respective atoms.

In order to develop a threshold to classify residues as buried or exposed, calculated ASA values need to be normalized based on corresponding reference values for given residue. This requires precomputed or predefined maximal accessible surface area (*MaxASA*) for all residues.

MaxASA is the maximal possible solvent accessible surface area for a given residue. MaxASA values are obtained from theoretical calculations of Gly-X-Gly tripeptides in water, where X is the residue of interest.[15] From ASA and MaxASA, the *relative solvent accessibility* (RSA) can be calculated by the formula:

$$RSA = \frac{ASA}{MaxASA}$$

Several scales for MaxASA have been published. eMap uses the most recent theoretical scale from Tien and coworkers.[15]

Relative solvent accessibility is calculated using the DSSP program developed by Kabsch and Sander.[16,17] In eMap, the RSA threshold chosen for exposed residues is:

$$\mathbf{RSA \geq 0.05}$$

as recommended by Tien and coworkers.[15] Residues with RSA greater than equal to 0.05 will be classified as exposed, those with lower RSA values will be classified as buried[2]. This threshold can be customized using the **ASA threshold** slider under Advanced options.

For more information on DSSP, please see `http://swift.cmbi.ru.nl/gv/dssp/DSSP_1.html`.

2.3.4 Graph Construction

When the user clicks the "Process" button, a pairwise distance matrix is constructed for the selected residues. The distance is calculated either between centers of mass of the side chains, or between their closest atoms. For standard protein residues, only side chain atoms are considered in the calculation. All atoms of automatically identified non-protein ET active moieties and user-specified custom fragments are considered in the distance calculations. From the distance matrix, an undirected weighted graph is constructed using NetworkX.

Penalty Functions

The next step is to cast the weights as modified distance-dependent penalty functions $P' = -log(\epsilon)$, where:

$$\epsilon = \alpha \exp\left[-\beta(R - R_{offset})\right]$$

α, β, and R_{offset} are hopping parameters, similar to the through-space tunneling penalty function in the Pathways model.[1] All subsequent calculations are performed using the modified penalty functions as edge weights. When using default hopping parameters ($\alpha = 1.0$,

[2] Note: RSA cannot be calculated for custom and user-specified residues, so such residues will always be classified as buried if RSA is used.

$\beta = 2.3$, $R_{offset} = 0.0$), the edge weights will be equal to the distances (multiplied by a prefactor of $2.3 \log(e) \approx 1$).

Distance thresholds and penalty function parameters can be modified under the Advanced Options tab described in Section 2.1.2.

Edge Pruning

One of two algorithms is used to prune the edges of the graph, which is specified by the Edge Pruning option under Graph Parameters in the Advanced options tab.

Percent-based algorithm

This algorithm considers only the smallest *Percent Edges* % of edges by weight per node, and then prunes based on the mean and standard deviation of the weights of the remaining edges.

Algorithm 1 Prune by Percent

```
procedure PRUNE(G(V,E), percent_edges, num_st_dev_edges, distance_cutoff)
    for v in V do
        for e in v.edges do
            if e.distance > distance_cutoff or e.weight > percentileweight(percent_edges) then
                G.remove(e)
            end if
        end for
        l̄ = mean_weight(v.edges)
        σ = st_dev_weight(v.edges)
        for e in v.edges do
            if e.weight > l̄ + num_st_dev_edges · σ then
                G.remove(e)
            end if
        end for
    end for
end procedure
```

Specify *Edge Pruning Algorithm* as **Percent** to use this algorithm. The *Edges per Vertex %*, *SD Cutoff*, *Distance Cutoff* sliders are used to control the *percent_edges*, *num_st_dev_edges*, and *distance_cutoff* parameters in Algorithm 1.

Degree-based algorithm

This algorithm greedily prunes the largest edges by weight of the graph until each node has at most *Max Degree* neighbors.

Algorithm 2 Prune by Degree

```
procedure PRUNE(G(V,E), max_degree, distance_cutoff)
    removal_candidates = []
    for e in E do
        if e['distance'] > distance_cutoff then
            G.remove(e)
        end if
    end for
    for v in V do
        if degree(v) > D then
            removal_candidates.append(v.edges)
        end if
    end for
    sort_by_weight_descending(removal_candidates)
    for e(u,v) in removal_candidates do
        if degree(u) > max_degree or degree(v) > max_degree then
            G.remove(e)
        end if
    end for
end procedure
```

Specify *Edge Pruning Algorithm* as **Degree** to use this algorithm. The *Max Degree* and *Distance Cutoff* sliders are used to control the *max_degree* and *distance_cutoff* parameters in Algorithm 2.

2.3.5 Shortest Paths

Once the graph is finished, the 2D graph image is rendered using the *neato* program in the Graphviz software, with all residues color-coded and classified as buried or exposed. The user is now able to calculate the shortest paths. There are two modes: "source only", or "specified target". For both modes, once the calculation is completed, the top 5 shortest pathways can be visualized within the NGL viewer using the checkboxes. The 2D graph image is updated as well, highlighting those residues involved in the shortest pathways. Additionally, information on all calculated pathways is presented in a table at the bottom of the webpage (Figs. 9, 10).

Source Only

When only a source is specified, Dijkstra's algorithm is used to find the shortest path between the specified source and every surface-exposed residue in the graph. The pathways are classified into branches based on the first surface-exposed residue reached along the pathway. Within a branch, the pathways are ranked according to their score.

For an example of how the branches are structured, refer to Fig. 9. In this example, W377, Y53, Y309, W492, and Y383 are surface-exposed

residues, and each of the pathways displayed is the shortest path to that residue from W400. For W324, the shortest path involves first going through W377, which itself is a surface-exposed residue. Thus this path is assigned to branch W377, and given the ID 1b, as it is the second shortest path in this branch. The shortest path to Y383 is given the ID 2a, and belongs to a different branch.

#	Source	1	2	3	4	5	6	7	Score
Branch: W377(A)									
1a	W400(A)	W377(A)							**7.23**
1b	W400(A)	W377(A)	Y53(A)						**17.51**
1c	W400(A)	W377(A)	Y309(A)						**21.22**
1d	W400(A)	W377(A)	W492(A)						**26.86**
Branch: Y383(A)									
2a	W400(A)	W334(A)	Y383(A)						**13.79**

Figure 9: Excerpt of pathways table generated when the analysis on PDB 1U3D is completed, with W400 specified as the source node.

Specified Target

When a source and a target are specified, a NetworkX procedure based on Yen's algorithm[5] is used to find the 5 shortest paths from source to target. Yen's algorithm exploits the idea that shortest paths are likely to share common steps, and is able to compute the k shortest paths between nodes in a graph with non-negative weights. The procedure first finds the shortest path, then finds the next 4 shortest deviations. The pathways are ranked according to their score.

#	Source	1	2	3	4	5	6	7	8	Score
Branch: W324(A)										
1a	FAD510(A)-2	W400(A)	W377(A)	W324(A)						**24.15**
1b	FAD510(A)-2	W385(A)	Y53(A)	W377(A)	W324(A)					**35.25**
1c	FAD510(A)-2	W400(A)	W334(A)	W379(A)	W324(A)					**36.37**
1d	FAD510(A)-2	W400(A)	W377(A)	W492(A)	W324(A)					**49.20**
1e	FAD510(A)-2	W385(A)	Y53(A)	Y309(A)	W377(A)	W324(A)				**50.67**
1f	FAD510(A)-2	W385(A)	Y53(A)	W377(A)	W492(A)	W324(A)				**60.30**

Figure 10: Pathways table generated when analysis on PDB 1U3D is completed, with FAD510-2 specified as the source node, and W324 as the target.

3 Protein Graph Mining: screening for common electron transfer pathways

3.1 Usage

This section describes different modes of use, functionality, analysis options, and visualizations available in the Protein Graph Mining tool. The overall workflow consists of three steps: file upload and parsing, mining, and the analysis of the results. Below we discuss each of the steps in details.

3.1.1 File Upload and Parsing

A user can upload a series of protein structures either by (i) specifying PDB IDs for the files to be fetched from the PDB database using the dialog window; (ii) uploading a .txt file containing PDB IDs separated by commas; and (iii) uploading a zip archive with PDB files to be parsed (Fig. 11).

Figure 11: The protein structures to be included into the analysis can be specified either by their PDB IDs explicitly (left window) or by uploading .txt or .zip files (right window).

A graph for each crystal structure is generated in the same way as in the single protein analysis case (Sec. 2.1.1).

3.1.2 Mining subgraph patterns

The next step after all specified protein structure files have been parsed is mining for common subgraph patterns. The algorithms and software used for mining are outlined in Sec. 3.2; here we focus on describing the relevant parameters and modes of use. Two modes of analysis are available. In the first, the user can screen a series of proteins for the existence of common electron transfer pathways (support number, minimum and maximum number of vertices have to be specified; see below). In the second, the user can search for a specific pattern (e.g. WWWY) in a series of proteins. Multiple choices for options specifying electron transfer active moieties, graph generation and mining parameters are available to the user at this step.

The **General** panel (Fig. 12) specifies the key parameters for the mining (minimum support, minimum and maximum number of vertices in the shared graph patterns, specific subgraph pattern), and the main parameters for the graph generation (distance options and aromatic amino-acids selection). Detailed descriptions of individual parameters are given below.

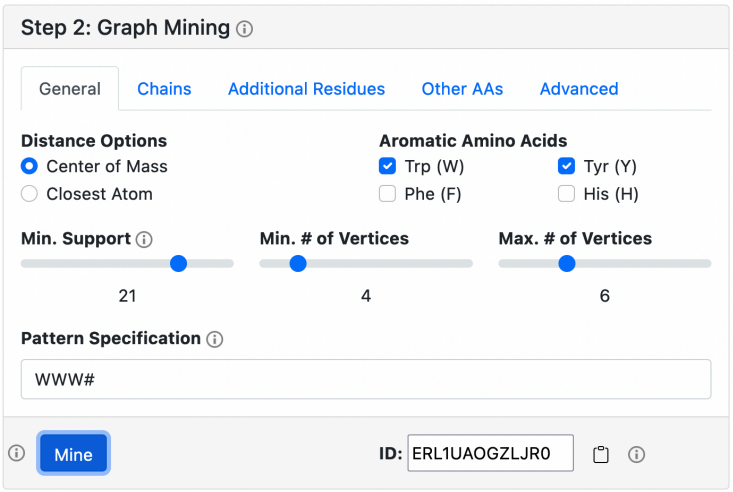

Figure 12: General panel specifying mining and graph generation options.

General

Distance Options: Distance between two ET active moieties can be calculated as the distance between the centers of mass of the side-chains, or as the distance between the closest atoms. Default: Center of Mass.

Aromatic Amino Acids: Standard amino acid residues to be included in the analysis. Default: Trp(W) and Tyr(Y).

Min. Support: Minimum number of PDBs that the mined subgraph patterns appear in. Increasing this parameter significantly reduces the computational time, while decreasing the value will likely result in more common subgraph patterns to be identified. Default: 8.

Min. # of Vertices: Minimum number of vertices in the mined subgraph patterns. Default: 4.

Max. # of Vertices: Maximum number of vertices in the mined subgraph patterns. Default: 6.

Pattern Specification: Specify subgraph pattern to be searched for in a series of PDB files. Amino acids are specified by their standard one-letter notation. "#" is used to specify non-protein ET active moieties (e.g. heme, flavin, etc.), "X" represents a special category

of residues which are considered interchangeable (see Sec. 3.2.1), and "*" represents a position where any residue can be found. Branching can be specified using a pseudo-smiles format (see Sec. 3.2.2). Default: None.

Note:
Min. Support, *Min. # of Vertices*, and *Max. # of Vertices* are ignored when the Pattern Specification is specified. Instead all possible subgraphs matching the pattern(s) will be found.

The General panel also shows a unique mining project **ID** issued once the Mine button is clicked (Fig. 12) that can be later used to load the project (Sec. 3.1.6).

Chains panel specifies the protein chains to be included in the analysis. Default: the first chain for each PDB file (most often the A chain).

Additional Residues panel specifies non-protein ET active moieties to be included in the graph. Default: all automatically identified non-protein ET active moieties on the selected chains are included.

Other AAs panel selects non-aromatic standard amino acid residues to be included in the analysis. Default: None.

Advanced panel allows the user to modify the advanced parameters used for mining (Fig. 13), building graphs, and defining the penalty function.

Mining

Edge thresholds: Defines a set of edge thresholds which categorize edges by their weights. See Sec. 3.2.1 for more details. Specify as a comma separated list of floats. Default: 12.0

Substitutions: Defines a set of amino acid residues which will be given the label 'X', and are considered interchangeable. See Sec. 3.2.1 for more details. Specify as a comma separated list of 1-character amino acid codes. Default: None.

Graph Parameters: These options are identical to the single protein analysis. See Sec. 2.1.2 for more details.

Penalty Function Parameters: These options are identical to the single protein analysis. See Sec. 2.1.2 for more details.

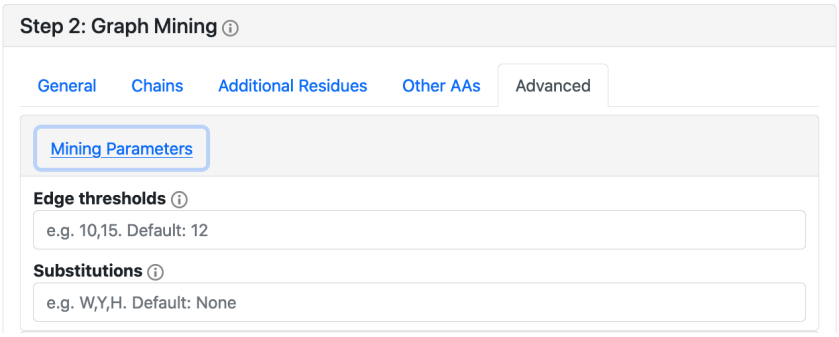

Figure 13: "Mining Parameters" section of the "Advanced" panel.

3.1.3 Analysis: Frequent Subgraph Patterns

The immediate result of a mining calculation is a set of *subgraph patterns*. These patterns are generic representations of shared pathways which were identified in the protein crystal structures. The mined subgraph patterns are assigned a unique ID which is displayed in the following format (Fig. 14):

{Unique index} : {String representation} | Support: {support number}

For each mined subgraph pattern, the panel shows its graph representation as well as a table summarizing the mining results (PDB IDs in which this pattern has been found along with the number of occurrences). The index shown next to the graph edge specifies the edge type (see Sec. 3.2.1).

A user can also download the overall mining report, the report for the specific subgraph pattern, as well as a PyeMap script that can be used to reproduce the mining results offline (Fig. 14).

Mining Report contains the list of parameters used for the mining and summary of the common subgraph patterns. An excerpt from a sample report is shown below.

```
Overview of all subgraphs:
Generated:
2022-01-19 11:54:59.336685
Graph Parameters:
{'sdef': None, 'dist_def': 'COM', 'rsa_thresh': 0.05,
'rd_thresh': 3.03, 'distance_cutoff': 20.0, 'percent_edges': 1.0,
 'edge_prune': 'DEGREE', 'num_st_dev_edges': 1.0,
  'coef_alpha': 1.0, 'exp_beta': 2.3, 'r_offset': 0.0}
Included residues:
['W', 'Y']
Mining parameters:
{'min_support': 8, 'graph_specification': '', 'min_num_vertices': 4,
 'max_num_vertices': 6}
Chains:
{'1EB7': ['A'], '1IQC': ['A'], '1NML': ['A'], '1RZ5': ['A'], '1RZ6': ['A'],
'1ZZH': ['A'], '2C1U': ['A'], '2C1V': ['A'], '2VHD': ['A'],
```

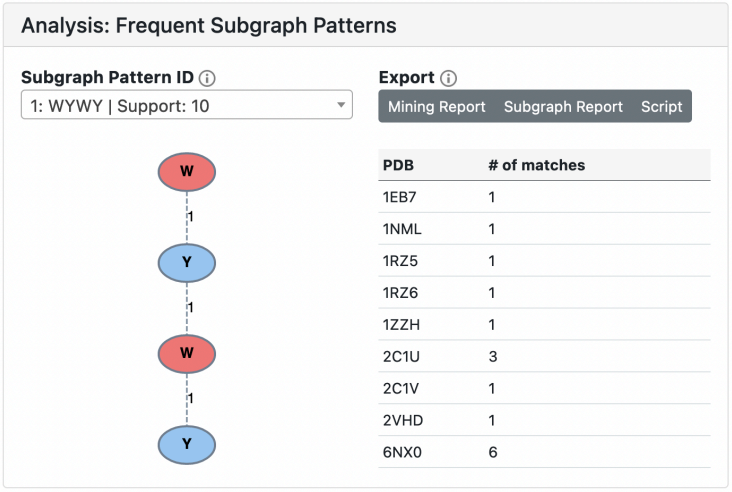

Figure 14: "Analysis: Frequent Subgraph Patterns" panel enabling inspection of the identified shared subgraph patterns and selection of a specific pattern for further clustering and analysis.

```
 '6NX0': ['A'], '6V59': ['A']}
Included non protein moieties:
{'1EB7': ['HEC401(A)', 'HEC402(A)'], '1IQC': ['HEC401(A)',
 'HEC402(A)'], '1NML': ['HEC401(A)', 'HEC402(A)'], '1RZ5':
  ['HEC401(A)', 'HEC402(A)'], '1RZ6': ['HEC401(A)', 'HEC402(A)'],
 '1ZZH': ['HEC802(A)', 'HEC803(A)'], '2C1U': ['HEC401(A)', 'HEC402(A)'], '2C1V':
['HEC401(A)', 'HEC402(A)'], '2VHD': ['HEC401(A)', 'HEC402(A)'],
'6NX0': ['HEC601(A)', 'HEC602(A)'], '6V59': ['HEC601(A)', 'HEC602(A)']}
Edge thresholds:
[12]
Node labels:
{'W': 2, 'Y': 3, 'X': 4, '#': 5}
Residue categories:
{2: 'W', 3: 'Y', 4: 'X', 5: '#'}

Subgraphs found:

ID:1_WYWY_10
Support:10
Where:['1EB7', '1NML', '1RZ5', '1RZ6', '1ZZH', '2C1U', '2C1V',
 '2VHD', '6NX0', '6V59']
Adjacency list:
W0:[Y1(1)]
Y1:[W0(1), W2(1)]
W2:[Y1(1), Y3(1)]
Y3:[W2(1)]

...
```

Relevant parameters listed in the report are explained below.

Graph parameters

sdef: surface definition

dist_def: distance definition

rsa_thresh: ASA threshold

rd_thresh: residue depth cutoff

distance_cutoff: distance cutoff

percent_edges: edges per vertex

edge_prune: edge pruning algorithm

num_st_dev_edges: SD cutoff

coef_alpha: penalty function α

exp_beta: penalty function β

r_offset: penalty function R_{offset}

For more details refer to the Sec. 2.1.2.

Mining parameters

min_support: Min. Support

graph_specification: Subgraph Pattern Specification

min_num_vertices: Min. # of Vertices

max_num_vertices: Max. # of Vertices

See Sec. 3.1.2 for the detailed definitions.

Each subgraph pattern found in the report includes the unique ID, support number (the number of PDB files in which the pattern has been found), the list of corresponding PDB IDs, and subgraphs' connectivity (adjacency lists).

Subgraph report provides a summary of the clustering results for the chosen subgraph pattern based on the structural or sequence similarity. An excerpt from a sample report is given below. Following the main

parameters of the subgraph pattern, the report lists subgraphs that have been clustered into the same group (Group 1 in the sample report) based on structurally similarity. For each protein subgraph (group member) the specific residues corresponding to the subgraph nodes in the PDB file are listed along with the graph connectivity (adjacency list) and the pairwise distances according to the distance definition used.

```
ID:1_WYWY_10
Support:10
Where:['1EB7', '1NML', '1RZ5', '1RZ6', '1ZZH', '2C1U', '2C1V', '2VHD', '6NX0', '6V59']
Adjacency list:
W0:[Y1(1)]
Y1:[W0(1), W2(1)]
W2:[Y1(1), Y3(1)]
Y3:[W2(1)]
22 subgraphs matching this pattern were found.
Graphs are classified based on structural similarity.

Group 1: 4 members
---------------
1RZ5(1)-1
Nodes
W266(A) Position in alignment:343
Y259(A) Position in alignment:336
W94(A) Position in alignment:145
Y211(A) Position in alignment:274
Adjacency list:
W266(A):[Y259(A)(8.41)]
Y259(A):[W266(A)(8.41), W94(A)(11.18)]
W94(A):[Y259(A)(11.18), Y211(A)(9.59)]
Y211(A):[W94(A)(9.59)]

2C1U(1)-1
Nodes
W280(A) Position in alignment:343
Y273(A) Position in alignment:336
W108(A) Position in alignment:145
Y225(A) Position in alignment:274
Adjacency list:
W280(A):[Y273(A)(8.37)]
Y273(A):[W280(A)(8.37), W108(A)(11.27)]
W108(A):[Y273(A)(11.27), Y225(A)(10.03)]
Y225(A):[W108(A)(10.03)]

...
```

Script is a Python script (pyemap_script.py) that can be used offline with the PyeMap software to reproduce the results shown on the page.

3.1.4 Analysis: Protein Subgraphs

The panel summarizes the protein subgraphs (i.e. the specific residues in the crystal structures) which match the chosen subgraph pattern Fig. 15).

Each protein subgraph for a chosen subgraph pattern has a unique ID,

which is displayed as:

{PDB ID}-{Unique Index}

For the selected subgraph pattern, the panel contains a table listing groups of protein subgraphs which have been clustered based on structural or sequence similarity (see Sec. 3.2.4). For each group, the size of the group and the corresponding PDB IDs are listed. The selected protein subgraph panel is visualized in 2D and 3D.

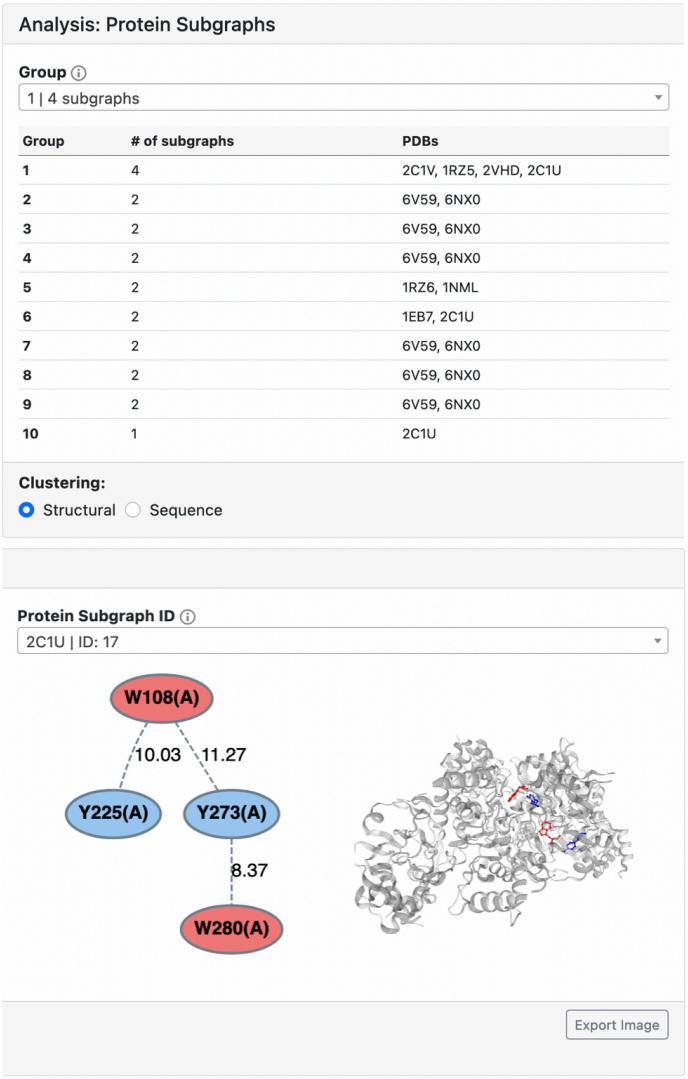

Figure 15: Protein Subgraphs panel summarizing clustering results including the information on the clustered groups of subgraphs and 3D subgraphs visualization.

3.1.5 Analysis: Multiple Sequence Alignment

The multiple sequence alignment panel shows the results of a multiple sequence alignment performed on the set of crystal structures by the MUSCLE[18] package. The row and the columns that correspond to the residues of the currently selected protein subgraph are highlighted in red. For example, for the protein subgraph visualized in (Fig. 15) that belongs to the 2C1U PDB file and is composed of residues Y225–W108–Y273–W280, the 2C1U row and the four columns corresponding to the four residues are highlighted in the alignment (Fig. 16).

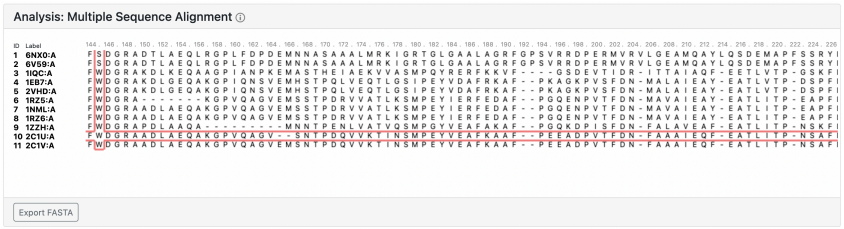

Figure 16: “Analysis: Multiple Sequence Alignment" panel.

3.1.6 Load Project

Using the Load Project page, the user can load a mining project (Fig. 17) using the project ID issued upon parsing of the initial input and shown in the General panel (Sec. 3.1.2).

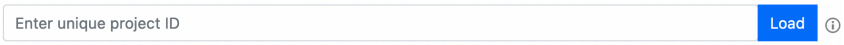

Figure 17: A user can upload existing project using Load Project page.

3.2 Algorithms

This section outlines the algorithms involved at each stage of the analysis, and the relevant parameters used to mine for common electron transfer pathways/motifs.

Protein graph mining with eMap involves 5 steps.

1. Generation of the protein graphs for each PDB in the analysis. This is done identically to the single protein analysis, and is discussed in Sec. 2.3.

2. Classification of the nodes and edges of each protein graph in order to generate a graph database.
3. Mining the graph database for shared patterns.
4. Graph matching to identify protein subgraphs.
5. Clustering protein subgraphs based on similarity to identify shared pathways/motifs.

3.2.1 Classification

The efficiency and descriptive power of graph mining is enhanced when the algorithms are able to distinguish between different types of nodes and edges. Graph mining in eMap relies on each node and edge in the graph database being assigned a numerical label which corresponds to its category. eMap offers some customization of these labels in order to broaden or narrow the search space.

Nodes

By default, each standard amino acid residue receives its own category, and all non-standard residues included in the analysis are labeled as 'NP' for non-protein (processed internally as '#'). One can specify a group of standard amino-acid residue types to be given the label 'X' (which is standard notation for unknown residue type), which enables these residues to be substituted for another in subgraph patterns. This is done by specifying a comma separated list of 1-character amino acid codes in the *Substitutions* field under the 'Mining Parameters' section in the Advanced options tab. See Fig. 18 for an example.

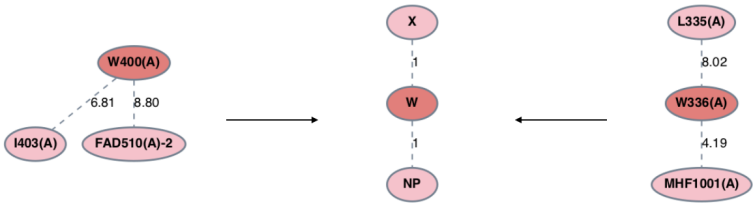

Figure 18: Example of two protein subgraphs belonging to the same subgraph pattern when Isoleucine (I) and leucine (L) are specified as substitutions.

Edges

By default, all edges are assigned the same numerical label of 1. The Edge thresholds field under the Mining Parameters tab in Advanced Settings allows one to specify a list edge weight thresholds in ascending order, where each value indicates a cutoff threshold for an edge category. See Fig. 19 below for an example.

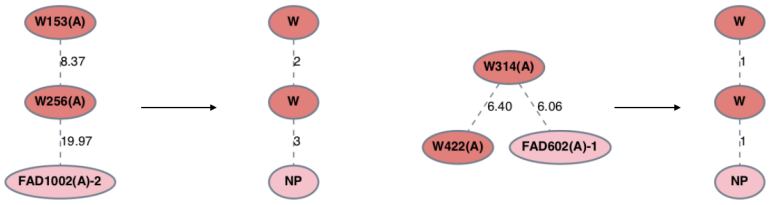

Figure 19: Example of 2 subgraph patterns and a corresponding protein subgraph for each pattern when the edge thresholds are set to "8,12".

3.2.2 Mining

The goal of subgraph mining is to identify graph structures which occur a significant number of times across a set of graphs. In the context of eMap, this means searching for shared pathways/motifs among a set of protein crystal structures. In eMap, users can either search for all patterns which appear in a specified number of PDBs (General Pattern Mining), or search for a specified group of patterns which match a string representation of the pattern of interest (Specific Pattern Mining).

In the graph mining literature, the frequency that a pattern appears in the set of input graphs is referred to as the **support** number for that pattern. In other words, if a pattern appears in 12/14 graphs, one would say it supports 12 graphs, or equivalently, has a support number of 12 (regardless of whether it appears multiple times within a given graph).

General Pattern Mining

In eMap, we use a Python implementation of the gSpan algorithm, one of the most efficient and popular approaches for graph mining.

The technical details are beyond the scope of this documentation (and can be found in the original gSpan reference), but here we emphasize a few of its key features.

- gSpan is a recursive algorithm which relies on the use of minimum depth first search (DFS) codes
- gSpan is a complete algorithm, so it will find all patterns which meet a minimum support number
- The performance of gSpan is greatly sped up by increasing the minimum support number, as this allows more aggressive pruning of candidate subgraphs

In eMap general graph mining is done by default when the *Pattern Specification* field is left blank. The "Min. Support", "Min # of Vertices",

and "Max # of Vertices" sliders are used to control what kinds of subgraph patterns will be identified.

Specific Pattern Mining

Instead of searching for all subgraphs which meet a minimum support number, one instead may be interested in finding protein subgraphs which match a previously known pattern. In this case, the problem is reduced to one of graph matching, and we simply search each PDB for subgraph isomorphisms using the NetworkX implementation of the VF2 algorithm (see Sec.3.2.3 for more details).

In eMap, the pattern(s) to search for are specified using the Pattern Specification field. The field accepts a string representation of a linear chain, where each character is one of the following:

- 1-character amino acid code for a standard residue
- X for unknown amino acid types
- # for a non-protein ET active moiety
- $*$ as a wildcard character

Branching can be specified using a syntax similar to the SMILES format. Each amino acid or special character described above must be separated by brackets (see example below), and bonding specification is dropped. The pysmiles package is used to generate the string representations of the graphs.

When edge thresholds are used (Sec 3.2.1), the search will be performed for all possible combinations of edges, and thus several subgraph patterns will be found for a set of residue types. If the $*$ wildcard character is used, subgraph pattern(s) will be found for each combination of each residue type replacing the $*$ placeholder character(s), including the special 'X' and '#' residue types. See Figs. 20–21 for examples.

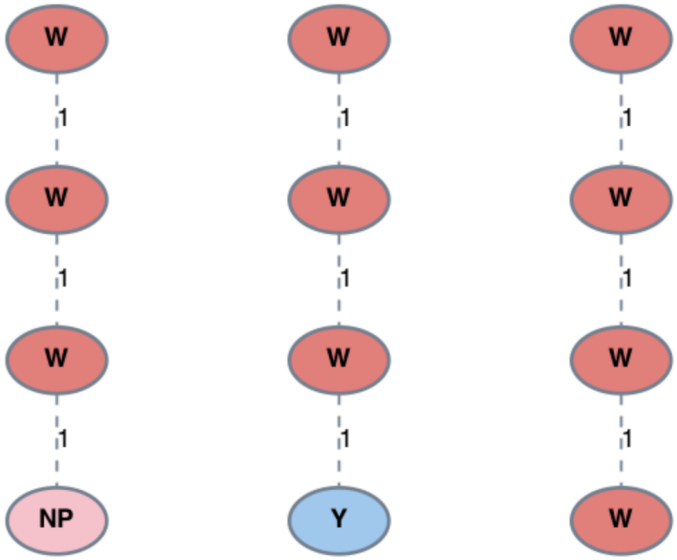

Figure 20: Examples of subgraph patterns identified using string 'WWW∗'.

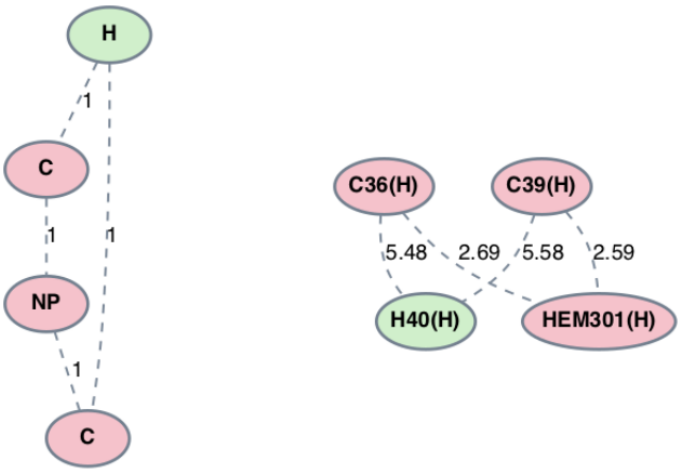

Figure 21: Subgraph pattern (left) and protein subgraph (right) identified using the pseudo-SMILES string '[*H*]1[*C*][#][*C*]1'.

Subgraph Patterns

The end result of either mining option is a set of *subgraph patterns*, each of which has a unique ID. In the **Subgraph Pattern ID** selection box, the ID is displayed as:

{Unique index} : {String representation} | Support: {support number}

e.g. 1 | WWW | Support: 18

The string representation for each pattern is a pseudo-SMILES string generated using the pysmiles package. Importantly, these strings can be used as inputs for the **Pattern Specification** field, as they correctly encode the structure of the graph using a syntax similar to the SMILES format.

3.2.3 Graph Matching

In order to identify the specific residues involved in subgraph patterns, we utilize the NetworkX implementation of the VF2 algorithm for graph matching. We refer the reader to their documentation for more details, but for the sake of clarity, we reiterate some definitions.

Let G=(N,E) be a graph with a set of nodes N and set of edges E:

If G'=(N',E') is a subgraph of G

- N' is a subset of N
- E' is a subset of E

If G'=(N',E') is isomorphic to G

- there exists a one-to-one mapping between N and N'
- there exists a one-to-one mapping between E and E'

For two graphs G(V,E) and H(V',E'), G and H are subgraph isomorphic if there exists a G'(V_0,E_0) such that:

- G' is a subgraph of G
- G' is isomorphic to H

Note that in some sources, the term *subgraph isomorphism* is reserved for when G' is a node- or edge-induced subgraph, and the term *monomorphism* is preferred for subgraphs which are not induced. The task of identifying all such G' is known as the *subgraph matching* problem, and we refer to individual G' as subgraph isomorphisms.

In eMap, for each PDB which supports the given subgraph pattern, we search for all subgraph isomorphisms within the protein graph. This gives us a set of *protein subgraphs*, which we can then cluster into groups based on similarity.

Each protein subgraph for a given subgraph pattern is assigned a unique ID, which is displayed in the Protein Subgraph ID selection box as:

{PDB ID}-{Unique Index}

e.g. 1u3d-2.

3.2.4 Clustering

Algorithm

eMap currenly enables two types of clustering: **structural** similarity, and **sequence** similarity, which both use the same underlying algorithm described below.

For a given subgraph pattern P, we have a set of protein subgraphs V which correspond to groups of specific residues in PDB structures which match pattern P. We then construct a supergraph G(V,E), where two protein subgraphs share an edge if and only if they are deemed sufficiently similar by the chosen metric. The resulting supergraph G (see Fig. 22) will be composed of one or multiple connected components, and each connected component corresponds to a cluster of similar protein subgraphs (see Fig. 23).

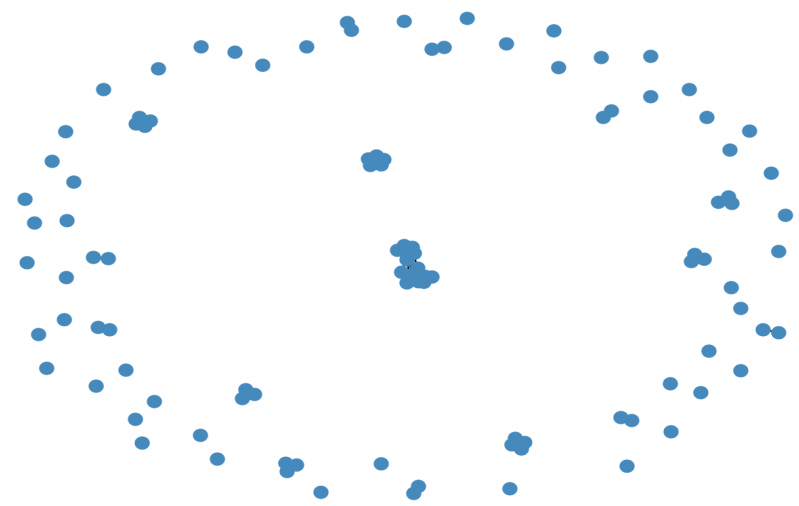

Figure 22: Example of supergraph generated by clustering algorithm. Each node corresponds to a protein subgraph belonging to the same subgraph pattern. Note the numerous connected components, which correspond to clusters of similar protein subgraphs.

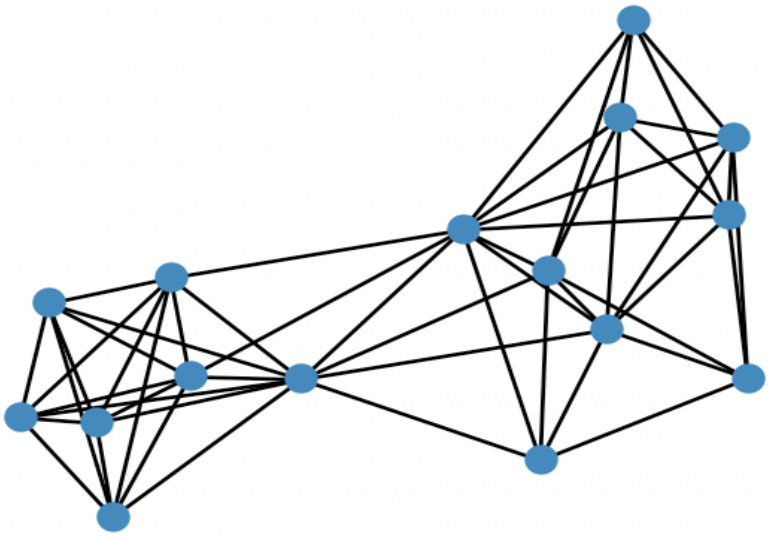

Figure 23: The largest cluster in Fig. 22. Each node corresponds to a protein subgraph.

Structural Similarity

The structural similarity between two protein subgraphs in eMap is computed by superimposing the two sets of atoms and computing the root mean squared distance (RMSD). However, atoms can sometimes be missing from crystal structures, and we would also like some flexibility to allow for substitutions. Starting from a one-to-one mapping between the residues, we make the following approximations to the true RMSD:

- Only the alpha carbon (CA) is considered for standard amino acid residues. If it is not present in the crystal structure, ∞ is returned.
- For non-standard amino acids, we use the first atom type both residues have in common. If no shared atom type is found, ∞ is returned.

The threshold used for determining whether two subgraphs are connected in the supergraph is 0.5 Å.

Sequence Similarity

Sequence similarity in eMap relies on a multiple sequence alignment, which will automatically be performed by the MUSCLE package. Starting from a one-to-one mapping between the residues, the sequence similarity between two protein subgraphs is simply defined as the differences in the residue numbers with respect to the multiple sequence alignment. For instance, TRP50 in one PDB and TRP200 in another PDB could have a difference of 0 if they are aligned by

the multiple sequence alignment. One important caveat is that non-protein ET active moieties are not considered in sequence similarity, only standard amino acids.

The threshold used for determining whether two subgraphs are connected in the supergraph is N, where N is the number of nodes comprising the subgraph pattern, which allows for slight misalignment.

4 Contact Us

Support and inquiries: emap.bu@gmail.com

Open an issue on GitHub: https://github.com/gayverjr/pyemap

5 Acknowledgments

The team is grateful for the support from Rafik B. Hariri Institute for Computing and Computational Science & Engineering and from The Molecular Sciences Software Institute under National Science Foundation grant ACI-1547580 (R.N.T.). Ruslan Tazhigulov also thanks Dr. Doaa Altarawy for valuable discussions.

6 Third-Party Applications, Components and License Information

- Biopython (Biopython permissive license and BSD-3-Clause)
- Bootstrap (MIT)
- DSSP (Boost Software License 1.0)
- Flask (BSD-3-Clause)
- Flask Table (BSD-3-Clause)
- jQuery (MIT)
- MSMS (MSLIB License)
- NetworkX (BSD-3-Clause)
- NGL Viewer (MIT)
- NumPy (BSD-3-Clause)
- PyGraphviz (BSD-3-Clause)
- RDKit (BSD-3-Clause)
- SciPy (BSD-3-Clause)
- Select2 (MIT)
- ViewerJS (MIT)
- MUSCLE (LICENSE)
- Redis (BSD-3-Clause)
- Celery (BSD-3-Clause)
- pysmiles (Apache 2.0)
- gSpan-mining (MIT)

References

[1] Onuchic, J. N.; Beratan, D. N.; Winkler, J. R.; Gray, H. B. *Annu. Rev. Biophys. Biomol. Struct.*, **1992**, *21*, 349–377.

[2] Beratan, D.; Onuchic, J.; Winkler, J.; Gray, H. *Science*, **1992**, *258*, 1740–1741.

[3] Rose, A.; Hildebrand, P. *Nucleic Acids Res.*, **2015**, *43*, W576–W579.

[4] Xifeng Yan; Jiawei Han. gspan: graph-based substructure pattern mining. in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp 721–724, 2002.

[5] Yen, J. Y. *Manag. Sci.*, **1971**, *17*, 712–716.

[6] Cock, P. J. A.; Antao, T.; Chang, J. T.; Chapman, B. A.; Cox, C. J.; Dalke, A.; Friedberg, I.; Hamelryck, T.; Kauff, F.; Wilczynski, B.; de Hoon, M. J. L. *Bioinformatics*, **2009**, *25*, 1422–1423.

[7] Hagberg, A.; Swart, P.; S Chult, D. *Exploring Network Structure, Dynamics, and Function Using NetworkX*. Technical report, Los Alamos National Lab (LANL), Los Alamos, NM, United States, 2008.

[8] Haynes, W. M. *CRC Handbook of Chemistry and Physics, 96th Edition (Internet Version)*. CRC Press/Taylor and Francis: Boca Raton, FL, United States, 2016.

[9] Rdkit: Open-source cheminformatics. http://www.rdkit.org, 2006.

[10] Chakravarty, S.; Varadarajan, R. *Structure*, **1999**, *7*, 723–732.

[11] Sanner, M. F.; Olson, A. J.; Spehner, J. C. *Biopolymers*, **1996**, *38*, 305–320.

[12] Song, J.; Tan, H.; Mahmood, K.; Law, R. H. P.; Buckle, A. M.; Webb, G. I.; Akutsu, T.; Whisstock, J. C. *PLoS ONE*, **2009**, *4*, e7072.

[13] Lee, B.; Richards, F. M. *J. Mol. Biol.*, **1971**, *55*, 379–400.

[14] Shrake, A.; Rupley, J. A. *J. Mol. Biol.*, **1973**, *79*, 351–371.

[15] Tien, M. Z.; Meyer, A. G.; Sydykova, D. K.; Spielman, S. J.; Wilke, C. O. *PLoS ONE*, **2013**, *8*, e80635.

[16] Kabsch, W.; Sander, C. *Biopolymers*, **1983**, *22*, 2577–2637.

[17] Touw, W. G.; Baakman, C.; Black, J.; te Beek, T. A.; Krieger, E.; Joosten, R. P.; Vriend, G. *Nucleic Acids Research*, **2014**, *43*(D1), D364–D368.

[18] Edgar, R. C. *BMC Bioinformatics*, **2004**, *5*(1), 113.