Dartmouth College Technical Report TR2018-862: PyMOL Plugin to Build Protein Structures Based on Natural TERM Overlaps

Noah Tiberius Paravicini
Advisor: Gevorg Grigoryan

November 21, 2018

Abstract

This project is a continuation of the Grigoryan Lab's exploration of TERMs. A TERM is a tertiary structural motif, which is a fragment of a protein that includes the secondary, tertiary, and quaternary environments around a certain residue. As displayed in past publications discussing TERMs, they are a useful way of decomposing proteins into smaller components that help in understanding design and prediction of protein structures.[1][2][3] The Grigoryan Lab developed a database that keeps track of naturally occurring overlaps between TERMs, which gives a user the information they would need to put these TERMs together into complex structures. These events led to the development of this project: The Protein Builder. The Protein Builder is a plugin for the molecular visualization software PyMOL that allows a user to build protein structures out of TERMs based on the database of overlaps. The Protein Builder is a Python program that implements a PyQt UI. Using the program's menu and PyMOL's interactive display of atomic structures, users can build structures from scratch, or save/load structures to build off of. The PyMOL client communicates with a server that contains the database, manages the state of a user's project session, and provides recommendations of TERMs to continue building with. The program is very useful to users with research needs, such as users wishing to use TERMs to learn more about protein structure & design or users that wish to design proteins that include certain residues or certain functionality.

1. Introduction

Tertiary motifs are a way of describing different pieces of a protein structure. Proteins are often understood in terms of their component parts. A protein's secondary structure describes the local structure and is driven by precise interatomic forces. In contrast, the domains of a protein describe general, high-level arrangements. There have been several proposals for components whose scale is in between that of secondary structure categorizations and domains[4][5]. Recently, the Grigoryan lab proposed a novel definition called TERtiary Motifs (TERMs), a way of decomposing a protein structure into discrete fragments based on residue-level contacts. Tertiary motifs allow us to divide up large protein structures into smaller pieces based on the residues that comprise them and their contacts, and then also visualize what structural components hold them together. Resources with more information on TERMs and their value can be found in the appendix.

Essentially, these TERMs are a useful and efficient way to cover structure space. Mackenzie et al. (2016) showed that more than 50% of natural proteins can be represented by roughly 600 different TERMs.[1] TERMs can allow a researcher to control the properties of a protein as they are theorizing it: for example, they can select for TERMs with a certain function (such as metal coordination or water binding[1]), and then build a protein structure around it based on other TERMs that function well with it. This is one of the key sources of motivation for the Protein Builder project: the lab wants to utilize the power of TERMs as a building block for proteins by creating software that lets users easily construct proteins from them.

In 2018, a PyMOL plugin that accomplishes the above goal was created. Though past efforts had gone into this project, producing a critical base from which the PyMOL plugin was developed from, the scope of this technical report will be on the PyMOL client and how it interacts with the server. Certain aspects of the server that were updated from the past in order to optimize the PyMOL client will also be in focus. The prior efforts made up to the development of the PyMOL client will be covered at length in Section 2.

2. Background and Prior Efforts

Prior to the development of the PyMOL client, a sizeable amount of effort had gone into defining common tertiary motifs as TERMs and creating means to work with them.

One of the most important steps in this process was the creation of a database that stored all of the TERMs and the information on which other TERMs they overlap with/can be connected to. Figure 1, shown below, displays an example of generated TERMs.

Figure 1: TERM Examples

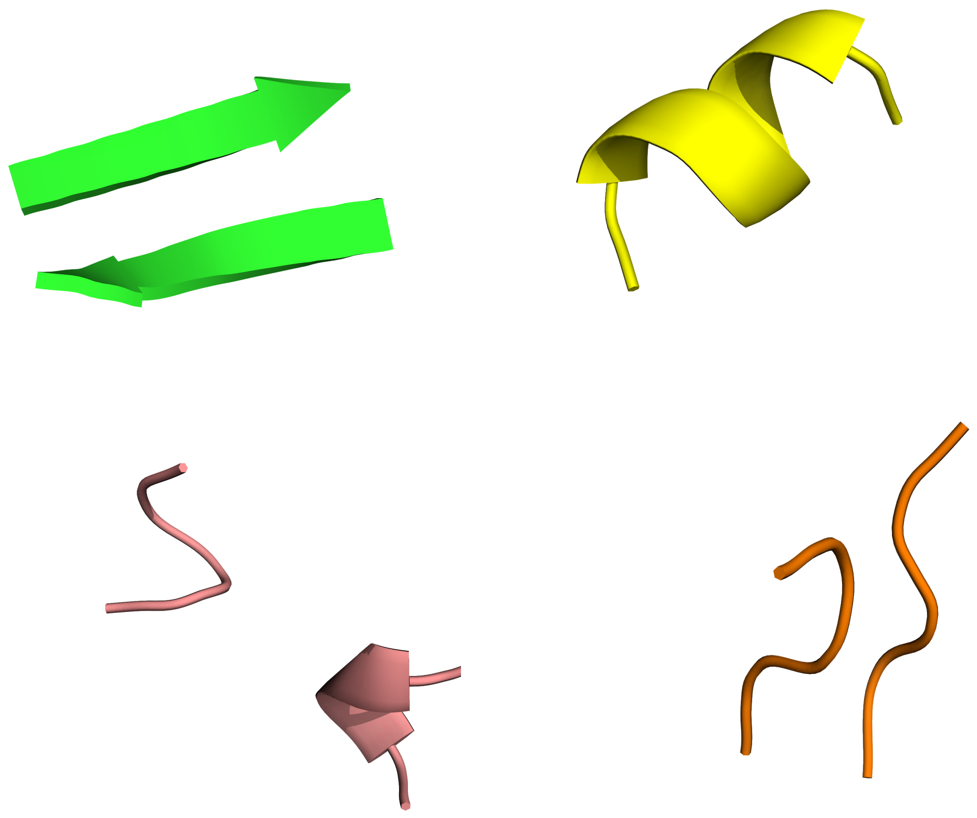


Figure 1 displays four examples of TERMs: two relatively simple ones (the top two), and two more complex ones (the bottom two). To put it simply, TERMs are decompositions of known protein structures; a full exploration into how TERMs were established can be found in Mackenzie et al. (2016), which is linked in the appendix. The TERMs were a novel idea, and the implementation of a database to format them and capitalize on their usefulness followed. The following pseudocode lays out how the TERM database and TERM overlaps are generated:

Generating the Overlaps

```
Collect the first $n$ TERMs, where 'first' means covering the most
structure-space*

Iterate over all possible pairs of TERMs

  For each pair:

    If the two TERMs occur in the same structure:

      If the TERMs have >= $x$ residues overlapping and the
      overlap can be extended by >= $y$ residues:*

        Align the TERMs (minimize RMSD between them)

        Combine the aligned TERMs into a single structure
        object

        Cluster the pair with Greedy Clustering (see below)

        For each cluster centroid:

          Extract the TERM pair from the extended structure

          Align the TERM pair with the centroid

          Store the transformation matrix and overlapping
          residues for the overlap in the database
```

** For the creation of the lab's database, these variables were set by Craig Mackenzie. In the lab's implementation, the first $n = 5000$ TERMs were used, and the residue parameters were set to $x = y = 1$.*

Greedy Clustering

Let n = the number of elements (coordinate sets), *nMax* = the maximum number of elements to brute force cluster (see below), and r = the RMSD cutoff**

```
While n > nMax:

  Select a set of nMax random sample elements

  For every element in the set:

    Brute Force Cluster the element

  For every element:

    If the element's RMSD to the centroid is <= r:

      Add the element to a new set

  Adjust the centroid to be closer to the average of all
  elements in the new set

  Remove the original set of nMax sample elements from the list
  of all elements

  Repeat

Brute Force Cluster the rest
```

*** Once again, Craig Mackenzie determined the value for the lab's implementation. RMSD was set to r = 1.0.*

<u>*Brute Force Clustering*</u>

```
For each remaining element:

  Declare the element the centroid

  Collect a set of all elements within r to it (the cluster)

Determine which element resulted in the largest cluster when
declared as the centroid

Remove the elements in this cluster
```

```
Repeat
```

Following this pseudocode generates a database of TERM overlaps. This database allows a user to identify relevant overlaps (by controlling the parameter *x* overlapping residues) and how to apply these overlaps (using the entered transformation matrix) to any composition of TERMs.

The creation of this database catalyzed the development of this project: now that these TERMs and the ways in which they can connect to others were clearly defined and accessible to a user, a prototype tool that could utilize this information was needed. The ideal model for the prototype was a server-client design: the database of TERMs & their overlaps would exist on a server, and a number of different clients could call upon the server to give them information on a certain TERM. The DALI Lab was tasked with creating a prototype, which was accomplished; they developed a server using C++ and a client using Unity. The server communicates information to the client on the position of a TERM, and the client displays it graphically. Upon clicking on a TERM, the server sends the client recommendations on which TERM can be added to it (based on its overlap info). These recommendations are displayed in thumbnails; clicking a thumbnail adds that TERM to the displayed structure.

Eventually, the lab made the decision to develop multiple client versions. The Unity client was to be for non-specialists in the fields of computational biology, genetics, etc., while a client for specialists would be developed on a different platform. The obvious choice for the latter was PyMOL, software designed for visualizing molecules. PyMOL can run community-generated plugins written in Python, making it extremely convenient to develop a client. Development for both clients have continued simultaneously, and both are currently in functional states.

3. Development

Generating a PyMOL Plugin:

The development of the PyMOL client began in May, 2018. Generally speaking, PyMOL plugins consist of external dialogue windows that work in tandem with the visual aspect of the software. Traditionally, these dialogue menus and windows were designed using the Python package tkinter, which provides a basic framework for developing graphical user interfaces (GUIs). However, PyMOL version 2.0, released in September 2017, brought about a significant change: PyQt replaced tkinter and other version-specific GUI packages as the universal package for generating dialogue boxes and menus.[6] PyQt is the name for the python binding of Qt, a powerful, C++ based all-purpose GUI framework that is utilized by many notable companies, such as LG, Panasonic, and AMD.[7] This meant that PyMOL plugins could now be designed through Qt, which was how Protein Builder was developed.

The PyMOL wiki page "Plugins Tutorial" (link in the appendix) provides instructions on how to write a python plugin file and use Qt Designer/Creator to design a UI for the plugin.[8] Unfortunately, at the time that this portion of the project was started, this page did not yet exist; consequently, the plugin file was originally based off of PyMOL's built-in plugins, found by exploring the source code and copying the lines pertaining to integrating the plugin into the software's plugin menu. From here, the process was very similar to what is outlined in the tutorial page: a UI file was designed using Qt software (Qt Creator/Qt Designer), and actions were bound to the UI buttons in the python file.[8] The specific design process and linking of the client with the server will be outlined below.

<u>Plugin Design:</u>

The client was designed to be a pop-up window that would communicate with the server and utilize the main PyMOL display area to visualize and interact with the structures. Upon running the plugin, the user will be met with a pop-up window that serves as the main interface between the user and the server (refer to Figure 2 for the current version of that window). The user can then interact with the buttons on the pop-up menu, as well as interact with the visualized structure as they normally would be able to in PyMOL. Clicking on a part of the structure will make a selection, which then allows the user to see which TERMs they can add to that part of the structure. The user can then save their structure to be reopened later.

The goal was to make the client as simplistic as possible, and allow the server to do the majority of the processing; the rationale behind doing so was that, in the future, it would make it easier to design clients on different platforms that all relied on the same server. As such, the client relies entirely on the server for information; the client only relays the user's choices to the server. For example, when a user makes a selection, the client sends a message to the server saying that a certain group of atoms was selected. The server then searches for which TERMs overlap with the group based on the selection mode, and then sends back a list of possible overlaps with accompanying thumbnails to the client. The client displays these thumbnails, and waits for the user to make a selection on one; the server is then alerted that a certain TERM has been added to the structure in the current session. This is then communicated back to the client, and the client displays that new TERM as part of the structure. Removing TERMs as well as undoing/redoing actions are all handled by the server as well.

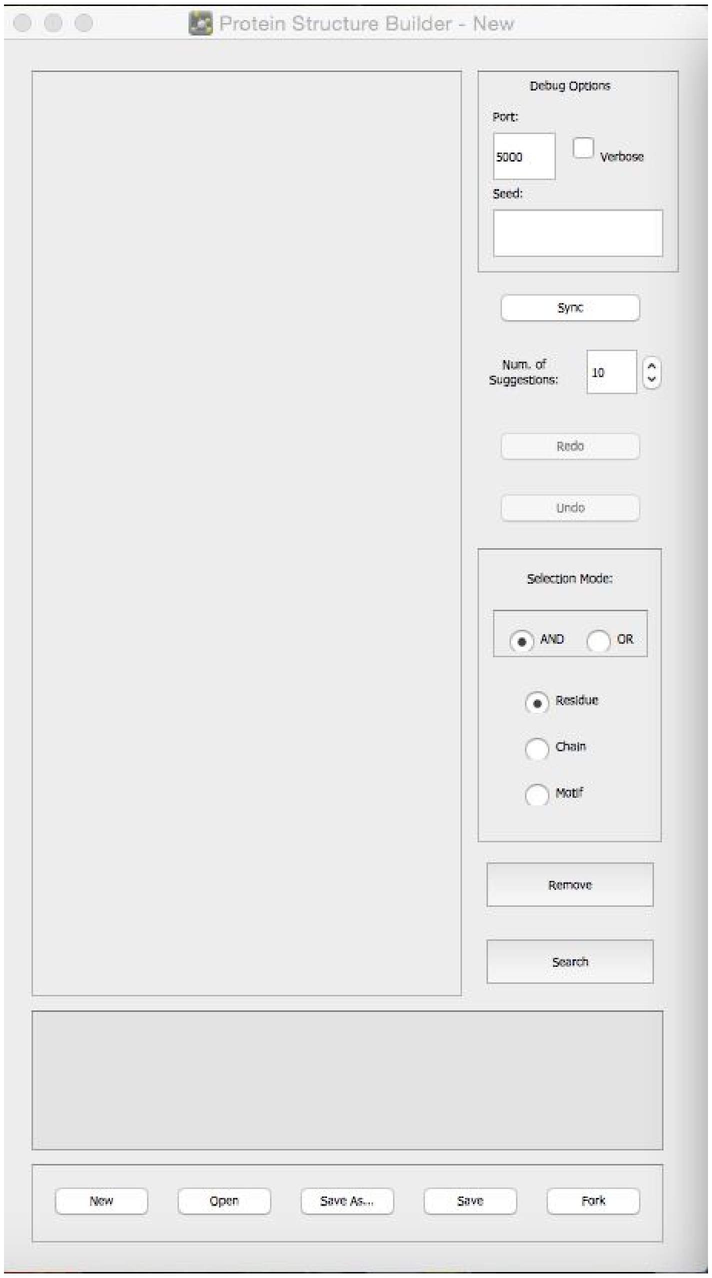
The server was designed to work with the client in sessions: when a new project is started on the client side, a new session is created on the server, and when an old project is loaded, the server would revisit the old session from that project. This would allow for data to be saved both on the client side and the server side. The save file was meant to be as simplistic as possible. Save files would be plaintext files consisting of a session ID, a date/time, and a list of every TERM in the structure. The list of the TERMs would be included so that a structure could be displayed without being connected to the server (however, it could not be altered without access to the server).

4. Current State and Technical Details

Current State:

Currently, the PyMOL client is capable of creating new structures, building them up based on naturally-occuring TERM overlaps, saving & loading structures, and undoing & redoing the placement of TERMs. All saving and loading is done through .pbs files, which are text files that contain a session ID, followed by the date and time of the most recent save, followed by the list of TERMs displayed. Refer to the figure on the next page for a breakdown of the client in its current state.

Figure 2: Client Menu



Key:

New: Creates a new .pbs file

Open: Opens a file explorer to open an existing file. Only allows selection of .pbs files.

Save As: Allows the current structure to be saved into a .pbs file with a custom name.

Save: If the structure being worked on has already been saved, re-save it under the same name.

Fork: Forks a copy of the current structure from the server.

Remove: Removes the selected TERM.

Search: Searches for TERMs that overlap with the selected TERM.

Selection Mode: Changes whether a mouse click on the structure selects a residue, chain, or motif.

Undo: Removes the most recently placed term; gives an error message if not possible.

Redo: Re-adds the most recently removed term; gives an error message if not possible.

Num. Suggestions: Changes the number of suggested overlaps shown at a time.

Sync: Syncs the state of the client with the state of the session on the server.

Debug Options: Seed refers to the RNG seed that the program uses, toggling Verbose prints everything the program does, and Port refers to the port the server and client connect through.

Technical Details:

The server and the client communicate to one another through strings that are sent as messages. Upon receiving a string, the server or client parses it to process it. The computing is done primarily on the server side; the client's primary functions are to send the server information to edit the session with and requests to acquire server data. The server takes queries from the client and searches the database for overlaps containing the query, and then compiles a list of suggestible overlaps sorted by frequency of occurrence. The server primarily sends messages containing a set of motifs for the client to parse and display (with the main exceptions of sending back the undo/redo number or the RNG seed after a client request). Below is a list of the messages the client can send.

Client Messages to Server
Get Overlapping Terms
Place Term
Remove Term
Undo
Redo
Check Undo/Redo Number
Sync
Fork Request
Set Atom Line Format
RNG Seed Request
Repopulate Suggestions
Search

The majority are self-explanatory or have been covered previously in this paper. The two not previously discussed are “Set Atom Line Format” and “Repopulate Suggestions.” The former tells the server that the client wants atom lines in PDB (Protein Data Bank) format. The latter requests a list of suggestions from the server following an undo/redo (so the user doesn't have to make another selection).

Usage Examples:

Figure 3: Example Structure

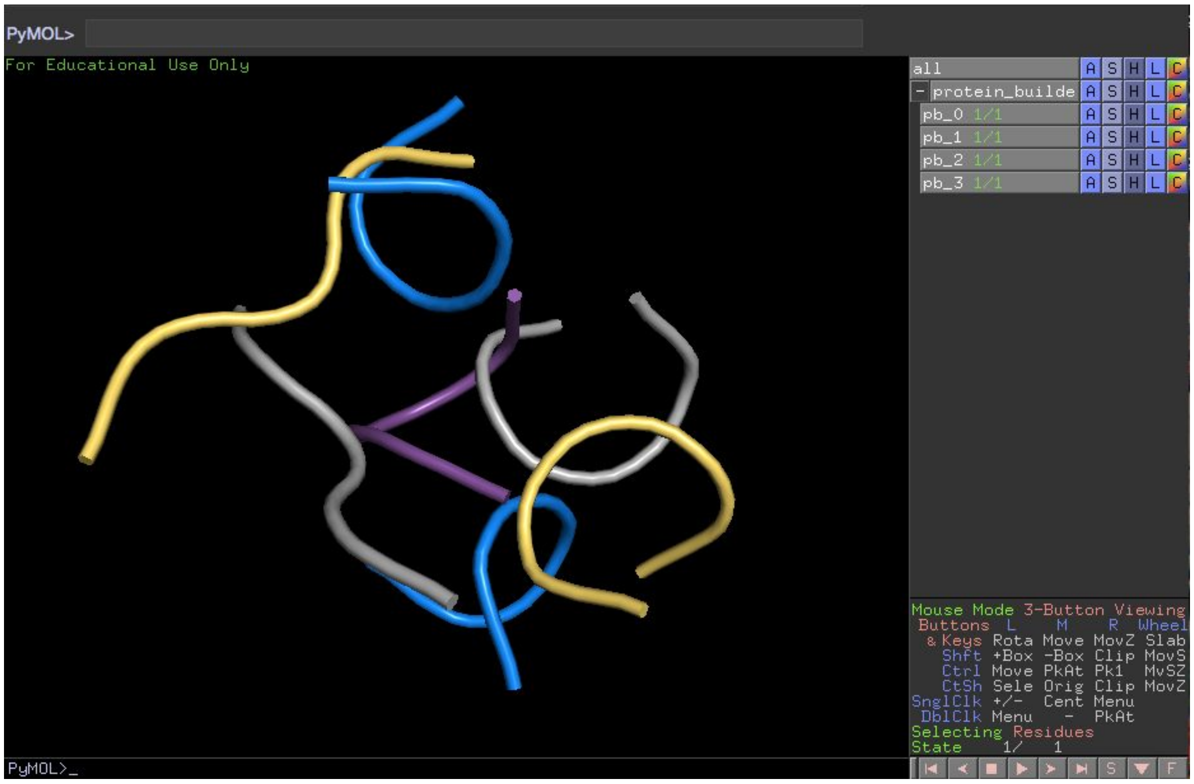


Figure 3 displays a very basic example of what the program can accomplish. The structure in the figure was built from scratch; we expect the program's ability to build up complex structures from nothing to be one of its most useful functions. Upon starting a "new" project, a list of suggested TERMs are given to the user, and the user can select any one of them to begin their structure; additionally, they could also search for a specific term. Once a structure is being built, the different selection modes give the user full control over how they want to build their structure. For example, if the user emphasizes the functionality of a certain residue, they can peruse all ways to build off of that residue; if they instead care about the inclusion of a certain chain, then they can select the chain and orient their building around it. The undo/redo functions are critical in this building process, as they give the user the freedom to explore every way of building off of a certain structure with no risk of making a mistake and having to start anew. The building process is designed to utilize the TERM database to give the user total control over the function and structure of their protein, allowing them to easily build proteins that reflect their visions. This could prove to be very valuable to users aiming to learn more about specific types of proteins or users looking to learn more about sequence-structure relationships.

Though the tool is not currently yet refined enough for this usage, one important purpose that the Protein Builder could serve in the future is the for the design of proteins in biomedical research. Zhou et al. (2018) discusses some of the current barriers to ideal computational protein design, and proposes a new framework for it involving TERMs.[9] Eventually, once certain limitations are surpassed, the Protein Builder or a tool similar to it could prove to be very practical in the medical field.

5. Future Steps

One notable improvement that could be made on the client would be the option to start the building process from a preset scaffold. These scaffolds could revolve around certain protein functions- as Mackenzie et al. (2016) discusses, certain TERMs appear specialized around different functions such as interface formation, metal coordination, and water binding.[1] Giving users preset scaffolds based on these functions would be very useful and allow them to more easily create complex proteins with desirable components.

Another future step that could be taken would be the development of fusion algorithms. As it stands, the client places these TERMs together in the correct positions in relation to one another, but the entire structure is still just a conglomerate of distinct parts instead of a whole, unified protein. Even in this state, the tool is still very useful in the design of proteins, but allowing the user to fuze the pieces into one cohesive, realistic protein would be the final step in making this the ideal tool for building up proteins from scratch. Doing so would require the development of fusion algorithms to conjoin overlapping pieces together.

Finally, there are multiple small, visual improvements that could be made to the client. Currently, there is no way to control what color placed or loaded TERMs will be; making this change would be minor, but could help users better focus on certain parts of their structures. Additionally, improvements are currently being made to the loading of suggestion thumbnails in the menu window; these will allow the user to get an effective preview of a TERM before placing it on their structure.

6. Acknowledgements

First and foremost, I would like to thank Jack Holland, a PhD student in the Grigoryan Lab. Jack has been instrumental in the success of this project and has made working on it an absolute pleasure. Throughout my time working on both the PyMOL client and the server, Jack was extremely helpful, managing which parts I worked on while still giving me plenty of independence to design things my own way. Additionally, he aided in the formulation of this paper: Jack provided me the TERMs for Figure 1 and the basis for the pseudocode following it, in addition to reading the paper and providing feedback while I was writing it.

Additionally, Bruce Zou '21 deserves a lot of credit for the creation of the PyMOL client. Bruce and I, under Jack's guidance, worked in tandem to create the plugin, and he is continuing to work on it still. Bruce is an extremely agreeable, hardworking student, and made life much easier for Jack and I throughout the development of the client.

I would like to acknowledge Craig Mackenzie and Jianfu Zhou for their extensive work on the topic of TERMs. They have done so much to establish TERMs as a useful concept and develop tools to utilize them, and are a large part of the reason we were able to create the Protein Builder. Additionally, their publications have really helped me understand the concept of TERMs and some of the goals of the lab.

Finally, I would like to thank Professor Grigoryan for agreeing to advise me on this project and letting me work in the lab full of brilliant people that he has created. Since the start of this project, I have learned so much about both proteins and software design, and it has made me very excited to explore more projects that apply computational skills to topics I am not very familiar with.

7. References

[1] Mackenzie, C. O., Zhou, J., & Grigoryan, G. (2016). Tertiary alphabet for the observable protein structural universe. *Proceedings of the National Academy of Sciences of the United States of America*, *113*(47), E7438-E7447.

[2] Mackenzie, C. O., & Grigoryan, G. (2017). Protein structural motifs in prediction and design. *Current opinion in structural biology*, *44*, 161-167.

[3] Zheng, F., & Grigoryan, G. (2017). Sequence statistics of tertiary structural motifs

reflect protein stability. *PloS one*, *12*(5), e0178272. doi:10.1371/journal.pone.0178272

[4] de Oliveira, S. H., Shi, J., & Deane, C. M. (2015). Building a better fragment library for de novo protein structure prediction. *PloS one*, *10*(4), e0123998. doi:10.1371/journal.pone.0123998

[5] Xie, Z. R., Chen, J., Zhao, Y., & Wu, Y. (2015). Decomposing the space of protein quaternary structures with the interface fragment pair library. *BMC bioinformatics*, *16*(1), 14. doi:10.1186/s12859-014-0437-4

[6] PyMOL v2.0 Release Notes. (2018, May 11). Retrieved from https://PyMOL.org/dokuwiki/?id=media:new2

[7] Company, T. Q. (2018). Qt | Cross-platform software development for embedded & desktop. Retrieved from https://www.qt.io/

[8] Plugins Tutorial. (2018, November 13). Retrieved from https://pymolwiki.org/index.php/Plugins_Tutorial

[9] Zhou, J., Panaitiu, A. E., & Grigoryan, G. (2018). A general-purpose protein design framework based on mining sequence-structure relationships in known protein structures. https://doi.org/10.1101/431635

8. Appendix

Additional reading on TERMs:

Mackenzie et al. (2016)
https://www.ncbi.nlm.nih.gov/pubmed/?term=Zhou+J%5BAuthor%5D+Mackenzie+Co%5BAuthor%5D

Mackenzie & Grigoryan (2017)
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5513761/

PyMOL Plugins Tutorial:
https://pymolwiki.org/index.php/Plugins_Tutorial