



NAVAL MEDICAL RESEARCH UNIT SAN ANTONIO

PROTEIN SEQUENCE-BASED DESIGN AND ANALYSIS SOFTWARE FOR THE DEVELOPMENT OF DIAGNOSIS TOOLS

STEVEN X. MOFFETT, PhD; DAVID J. LEMON, PhD; APRIL A. FORD, BS; HOLLY C. MAY, PhD;
EUN Y. HUH, MS; YOON Y. HWANG, PhD

MAXILLOFACIAL INJURY AND DISEASE
CRANIOFACIAL HEALTH AND RESTORATIVE MEDICINE

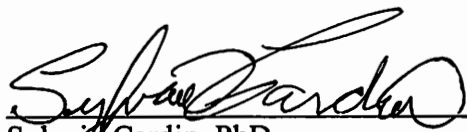
NAMRU-SA REPORT # 2020-607

Approved for public release; distribution is unlimited

DECLARATION OF INTEREST

The views expressed in this article are those of the authors and do not necessarily reflect the official policy or position of the Department of the Navy, Department of Defense, nor the U.S. Government. This work was funded by Defense Health Agency using work unit number G1716. Authors are employees of the U.S. Government. This work was prepared as part of their official duties. Title 17 USC §105 provides that 'copyright protection under this title is not available for any work of the US Government.' Title 17 USC §101 defines a US Government work as a work prepared by a military service member or employee of the US Government as part of that person's official duties.

REVIEWED AND APPROVED BY:



Sylvain Cardin, PhD
Chair, Scientific Review Board
Chief Science Director
Naval Medical Research Unit San Antonio
3650 Chambers Pass, BLDG 3610
Fort Sam Houston, TX 78234-6315

06/06/2021
Date



CAPT Gerald T. DeLong MSC, USN
Commanding Officer
Naval Medical Research Unit San Antonio
3650 Chambers Pass, BLDG 3610
Fort Sam Houston, TX 78234-6315

06/16/2021
Date

TABLE OF CONTENTS

ABBREVIATIONS	4
EXECUTIVE SUMMARY	5
INTRODUCTION	6
MATERIALS AND METHODS	8
RESULTS	10
DISCUSSION	11
MILITARY SIGNIFICANCE	12
REFERENCES	14
FIGURES	15
SOURCE CODE.....	16

ABBREVIATIONS

PUnQT	Peptide Uniqueness Quantification Tool
PLA ₂	Phospholipase A ₂
SAW	Surface Acoustic Wave
TBI	Traumatic Brain Injury

EXECUTIVE SUMMARY

Background: Forward deployments to austere areas introduce unique types of injuries. Further, conventional injuries can be more difficult to treat due to reduced access to therapeutic medicaments or diagnostic tools. Unique biomarkers released in response to injury and illness are crucial components of the development of diagnostic devices. Their identification, using specific probes, can guide optimal treatment courses. Most biochemically significant proteins in taxonomically close organisms have similar amino acid sequences, secondary structures, and tertiary structures. Therefore, any probe used to detect a protein biomarker of injury or illness must be able to bind to a unique site on the biomarker protein. In order to identify high-affinity binding agents to these unique biomarker active sites, small peptides can be synthesized to serve as an epitope-mimicking target for library screening methods such as phage display. When a protein must be identified preferentially over another closely-related protein, or group of proteins, the synthesized peptide should represent the most unique region of the biomarker's amino acid sequence where binding can occur. Selecting and analyzing these unique sequences manually with database queries and local software tools are time-consuming, labor intensive, and error-prone.

Objective: To combat the frailties of manual sequence-querying and selection, we designed an automated program that retrieves multiple protein amino acid sequences, aligns them, identifies unique amino acid subsequences, and generates a graphic to verify sequence correspondence to a region likely on to be the protein's exterior surface, all presented in a simple user interface.

Methods: A script in the Python programming language was designed to perform the objectives above. The `Biopython` and `Matplotlib` libraries were used for analysis, and the `Tkinter` library was used for the graphical user interface. The `ClustalOmegaCommandline` library was used for controlling the Clustal Omega program to perform sequence alignments locally.

Results: A sample run of the program compares homologous venom components of two related snake species. A user-requested subsequence length of 50 amino acids for each snake's venom component generates a unique target sequence for each component, with visual data showing the relative hydrophobicity for each species' target sequence along its macromolecule length.

Conclusions: We offer the **Peptide Uniqueness Quantification Tool (PUnQT)**, a Python program that automates the generation of unique peptide targets for high-throughput protein target selection in a rapid and straightforward manner while minimizing possible user error.

INTRODUCTION

Austere deployment environments typically lack the expensive and large but accurate and precise diagnostic tools found in modern hospital settings. Accurate and rapid diagnosis of injury or disease is crucial for selection of the proper treatment course and paramount to implementing successful treatments. The creation of a diagnostic tool that is portable, inexpensive, rapid, and accurate will vastly improve the ability of Corpsmen to select the proper treatment course in field operations. In the area of diagnostic device development, two promising technologies are the surface acoustic wave (SAW) and optical sensor platforms for detecting protein-peptide interactions (Mujahid, Afzal, & Dickert, 2019; Mukundan et al.). Although they are not fieldable at the time of writing, surface acoustic wave devices are currently being developed for use as a small handheld diagnostic device ("Aviana Molecular Technologies," 2020). While the tool described herein was developed for these potential field devices, it would be well-suited to any other diagnostic method that requires generating peptide molecular targets.

Protein-protein and protein-peptide interactions are the basis of many biological processes. While biological studies of protein interactions can employ isolated, full proteins (Cossins & Lawson, 2015), the full proteins of interest may have several binding sites shared with other, irrelevant proteins. Alternatively, a sequence-specific span of the protein can be synthesized *de novo* to serve as an experimental target for experiments involving specific protein-protein interactions. These shorter, representative peptides could provide the protein macromolecules' active sites or other clinically-relevant features of interest (Titus, Kay, Glaser, & Hwang, 2017). Use of SAW devices, optical sensor devices, or current detection modalities such as lateral-flow strips or enzyme-linked immunoassays (ELISAs), would require highly specific and selective probe molecules which are usually found through a massive screening procedure.

Previously, our group used phage display panning methodology to yield phages that bind with high affinity and specificity to sites on the venom peptides of several snake species (Titus et al., 2017). Phage display allows researchers to rapidly identify short peptides that will bind to biological targets (Fralick, Chadha-Mohanty, & Li, 2008). The phages that bind to biological targets can be used to identify biomarkers that exist only in a single species of snake, which will expedite the differential diagnosis of snake bites and guide subsequent treatment. While the use of whole venoms as phage display targets would expose the phages to many irrelevant and/or

nonspecific binding sites shared among all snake venoms, a synthesized peptide will optimize identification of phages that bind to unique venom components of differing snake species. To act as an appropriate species-specific binding target, the chosen peptide subsequence must be part of a protein that is at a sufficient concentration within a particular venom, and be specific to the envenoming species. For this project, we analyzed the major venom components of North American venomous snakes; phospholipase A₂, serine proteinase, and metalloproteinase (Engmark et al., 2017; Tasoulis & Isbister, 2017). The toxic component that has the most sequence uniqueness between species will serve as the protein macromolecule from which the unique peptide subsequences will be chosen.

Manual selection of viable peptide sequences to find target sequences in the above screening procedures involved time-consuming steps including extensive database querying, complex software tools, and secondary analysis of the peptide sequence, all of which facilitate human error. Automating the selection of protein targets by *in silico* analysis will lead to higher throughput of biomarker candidates and will generate a specific diagnostic test more rapidly than selection of peptide targets manually. An alternative for manual selection of viable peptide sequences is a single software tool that automates peptide selection and analysis, curtailing the time costs and minimizing error. To properly identify a unique amino acid subsequence within the protein to act as a peptide target, such a tool must perform five steps: 1) retrieve the protein sequences of each snake species' homologous venom components, 2) align these protein sequences to one another, 3) identify the amino acid subsequence that is most unique to each homologous protein sequence, 4) allow a researcher to verify that the subsequence corresponds to a hydrophilic region on the macromolecule (i.e., a region likely to be accessible to interacting molecules within an aqueous environment) by providing a graphical representation of hydrophobicity/hydrophilicity across the sequence, and 5) sum the amount of residues in the sequence that affect lab-based peptide synthesis.

To address all of these concerns, we developed a tool within the Python programming environment to automatically generate a maximally-unique peptide sequence which requires only user input of the UniProt identification numbers of the proteins of interest. This software, **Peptide Uniqueness Quantification Tool (PUnQT)**, was written to assist with the selection of peptide targets for discriminating between species-specific snake venom proteins. However, it

has broad applicability to situations in which individual, unique subsequences are needed within a group of homologous protein sequences.

MATERIALS AND METHODS

Materials

Hardware. PC with Windows 10 Enterprise (64-bit), Intel Core i5-8350U CPU @ 1.70 GHz, and 8.00 GB memory. However, PUnQT should be usable with any operating system/hardware which includes the following parameters: (1) internet access to query UniProt's online database and (2) the software suite used.

Software. Python programming environment (version 3.6) ("Python,") with its built-in `urllib` module, the Biopython Python library (version 1.72) ("Clustal Omega,") including Biopython's `ExPASy` module, Biopython's `SeqIO` module, and Biopython's `ClustalOmegaCommandline` module, the Clustal Omega multiple sequence alignment tool (version 1.2.2, Windows 64-bit) ("Matplotlib,") Matplotlib Python library (version 3.0.3) (Kyte & Doolittle, 1982), and Tkinter GUI library (Tkinter, 2020).

Procedures

Sequence querying - The script first prompts the user to provide the number of protein sequences they wish to analyze to reveal a unique short protein target peptide, followed by the UniProt protein identifiers for each sequence. This portion of the script uses the Biopython library's `ExPASy` module to query FASTA-format files from the UniProt protein database. The FASTA files are converted by the Biopython library's `SeqIO` module into a sequence (`Seq`) object that includes its protein identification information and protein sequence.

Sequence alignment - UniProt uses the Clustal Omega multiple sequence alignment software to perform its alignments. As an alternative to Biopython's built-in alignment tool, the library provides the `ClustalOmegaCommandline` module, a wrapper for the Clustal Omega software. The UniProt database does not allow programmatic access to sequence alignment queries or retrievals. While the Biopython library contains a pairwise sequence alignment module, it does not contain a multiple sequence alignment module, making it unsuitable for cases with more than

two query protein sequences. Further, Biopython's pairwise alignment tool uses a different alignment algorithm than UniProt's, creating an inconsistency between the tool and the results obtained if a user prepares the sequence data by using the UniProt website and querying an alignment locally using Biopython, making a tool like PUnQT even more necessary for consistent scientific data. Using the `ClustalOmegaCommandline` module, PUnQT loads all user-provided proteins' sequences into the alignment tool. Clustal Omega returns an `alignment` object with all sequences aligned with each other, converting missing spans of amino acids from the unaligned sequences to dashes (-).

Amino acid subsequence generation – PUnQT prompts the user to provide the desired peptide sequence length in a yellow-highlighted field, raising an error if the subsequence is longer than the query protein sequences. The aligned sequences are then passed to the `uniqueness_marker` function. This function generates a uniqueness score for each amino acid at the amino acid's position within the aligned sequence. PUnQT iterates over each aligned sequence's length one amino acid at a time, comparing the present amino acid with the other amino acid sequences in the same position of the alignment. If the amino acid is different from one of the other sequences' amino acids, PUnQT adds 1 to the index of that amino acid's position on the query sequence; a completely unique amino acid at a given position in a group of n query proteins, therefore, will have a score of $n-1$ at that position. If the amino acid of the aligned version of the query sequence is a (-) character, denoting a gap in alignment, the tool assigns a 0 to that position and moves to the next one.

By the end of the sequence, the number at each amino acid position denotes the number of sequences from which that amino acid is unique. The script then repeats this for all other query protein sequences.

The sequence scores and the user-specified peptide length are then passed to the `uniqueness_adder` function. The tool creates a sliding window of user-defined peptide length. Within the window, the sum of all individual uniqueness scores is generated. This window is scanned across each sequence, summing the uniqueness scores at each point, and locating the position of the window with the maximal sum over the entire sequence (comprising the sequence with the highest average uniqueness of user-defined length). The index of this window is then

returned to the main function. The main function then displays the starting index value of the most unique amino acid subsequence for each protein, along with the peptide subsequence itself.

Hydrophilicity verification graph/cysteine-methionine count – Groups of hydrophilic residues tend to have greater accessibility to water in solution (Kyte & Doolittle, 1982). In PUnQT, the aqueous environment accessibility is predicted from Stephen White's Experimentally Determined Hydrophobicity Scale (White, 2011). The user is prompted to give a hydrophobicity window size, expressed in number of amino acids. This user-defined value and the protein sequences are then passed to the hydrophobicity function. Similar to the `uniqueness_adder` function, this function slides the hydrophobicity window across the sequence, reading the subsequences and summing the hydrophobicity values of each amino acid within the window, which provides a numerical value for total free energy. The total free energy values of each species-specific target sequence are stored in a dictionary. If the sequence reader iterates over a dash (-), or other non-amino acid character, nothing is added to the total free energy sum, and the next character is considered. The `hydrophobicity` function returns a dictionary data structure in which the sequences are the keys, and a list of the windowed sums is each key's value.

This dictionary is then passed to the `plotter` function, which plots each sequence's total free energy measurements as a function of each sequence's index number. The span of amino acids corresponding to the species-specific target sequence is emphasized as a highlighted, yellowed region of the plot. In this way, the user can verify that the index value displayed for the unique subsequence corresponds to an area of higher amounts of total free energy.

The `hydrophobicity` function also displays a count of methionine and cysteine residues within the sequence. The count of each of these hydrophobic residues affect peptide synthesis due to their oxidation and/or sulfide bridge-forming properties, with high amounts of both residues indicating a more difficult synthesis process (White, 2011).

RESULTS

Example Analysis: Snake Phospholipase A2: The snake phospholipase A2 sequence was chosen for illustrative purposes, to both show the conservation of most snake species' destructive venom sequence and to illustrate the discriminatory powers of the tool. Comparisons with serine or metalloproteases generate similar results. Executing the program, the user is prompted to give

the number of sequences to compare, then to insert UniProt identification numbers for each sequence (Figure 1a). In this example, the UniProt identifiers for phospholipase A₂ (PLA₂) in Hardwick's Spine-Bellied sea snake (Q8UW08) and in the Southern Copperhead snake (A0A194AQ80) are used as sample user-provided data. These two species have similar PLA₂ enzymes, but they differ in their sequence enough to demonstrate the tool's capabilities.

PUnQT then queries the UniProt database and aligns the sequences using the Clustal Omega program. Once the aligned PLA₂ sequences are returned, the program refers these aligned sequences to the `uniqueness_marker` function. It also saves the aligned sequences in a text file for later viewing. The program then prompts the user to specify the sequence length of the desired peptide for future phage display experiments, using the value in the `uniqueness_adder` function. In this example, the chosen sequence length is 50 amino acids. The program outputs the maximally unique species-specific target sequence for each UniProt identifier, as well as its starting index number within the query sequence (Figure 1b).

Finally, the program prompts the user to specify the window size for the hydrophilicity plot. Usually, this window size is the same size as the subsequence length but the option is left to the user to show coarser (larger window size) or finer (smaller window size) resolution throughout the query protein in cases of a larger synthesized peptide or a specific desired positioning relative to hydrophilic areas. In this example, the chosen resolution of the hydrophobic window is five amino acids (Figure 1c).

The program uses the `plotter` function to return a graph showing each protein's hydrophilicity summation over the entire protein at the user-defined resolution, highlighting the area of the greatest uniqueness in yellow. A .png file of this graph is also saved for later review. In this example, the most unique span of the Copperhead snake sequence is in a region of high total free energy, implying a higher likelihood of this span of peptides being in an aqueous-accessible region of the protein than the most unique sequence of the Hardwick's sea snake. Further, the methionine/cysteine count is shown (Figure 1d).

DISCUSSION

In this report, we offer PUnQT, a graphical user interface-directed Python program that automates the 1) retrieval of amino acid sequences of multiple proteins, 2) alignment of all protein sequences to one another, 3) identification of the amino acid subsequence most unique to

each protein within a group of homologous species, 4) generation of a graphic to verify that the sequence corresponds to a region likely to be exposed to the surrounding aqueous environment, and 5) provides a count of amino acids with special implications for peptide synthesis. PUnQT's automation eliminates time-consuming and user error-prone tasks of website querying, and generates identical alignment data to that on the UniProt database website. The tool generates its alignments within seconds locally, rather than requiring several minutes when queried on the website. This tool assists the well-developed technique of protein synthesis to generate high-throughput protein targets, and when used to analyze several different homologous proteins, the likelihood of choosing an optimal, or close to optimal, peptide target sequence is high.

The representative peptide derived from PUnQT can then be used in conjunction with high-throughput selective processes, such as phage display, to yield high-affinity complementary peptides (Fralick et al., 2008; Titus et al., 2017). Complementary peptides can isolate epitopes or mimotopes for targeting pathogens (Gorr, Flory, & Schumacher, 2019), manufacturing biomarker probes (Hwang et al., 2017), and inhibiting toxic proteins (Titus et al., 2017).

PUnQT provides a solid foundation to improve target peptides; however, successfully predicting a protein's peptide sequence that can serve as a biological target depends upon more attributes than hydrophobicity, methionine-cysteine counts, and non-identical amino acid residues. The possibility of salt-bridge interactions, secondary structures as a result of disulfide binding within the peptide, or other confounding factors are possible, but cannot be predicted with this tool alone and require further scrutiny by researchers after generating their unique sequences. Gathering and interpreting as much information about the target protein as possible would result in more accurate predictions of successful peptide targets. Informed choice when choosing target peptides will yield peptide sequences that will reliably act as biological targets and promote easier, higher throughput, and less costly immunological studies.

MILITARY SIGNIFICANCE

The military is interested in diagnostic biomarkers for wound healing, traumatic brain injury (TBI), and many others (Beidler et al., 2008; Hahm, Glaser, & Elster, 2011; Marion, Curley, Schwab, & Hicks, 2011). The PUnQT tool can assist in identifying the most unique span within a set of closely-related endogenous human proteins biomarkers of wound healing or TBI.

For example, identification of highly-specific epitopes using the PUnQT tool may accelerate the identification of military biosensor-based diagnostic targets, improving treatment outcomes and increasing warfighter readiness. Current diagnostics in controlled medical environments use ELISAs for diagnostic purposes; the PUnQT tool can be used presently to generate peptide targets for use with ELISAs as an alternative to antibody-based molecular probes.

REFERENCES

- Aviana Molecular Technologies. (2020). Retrieved from <https://avianamolecular.com/product/>
- Beidler, S. K., Douillet, C. D., Berndt, D. F., Keagy, B. A., Rich, P. B., & Marston, W. A. (2008). Multiplexed analysis of matrix metalloproteinases in leg ulcer tissue of patients with chronic venous insufficiency before and after compression therapy. *Wound Repair Regen*, 16(5), 642-648. doi:10.1111/j.1524-475X.2008.00415.x
- Clustal Omega. Retrieved from <http://www.clustal.org/omega/>
- Cossins, B., & Lawson, A. (2015). Small molecule targeting of protein-protein interactions through allosteric modulation of dynamics. *Molecules*, 20(9), 16435-16445.
- Engmark, M., Lomonte, B., Gutiérrez, J. M., Laustsen, A. H., De Masi, F., Andersen, M. R., & Lund, O. (2017). Cross-recognition of a pit viper (Crotalinae) polyspecific antivenom explored through high-density peptide microarray epitope mapping. *PLOS Neglected Tropical Diseases*, 11(7), e0005768. doi:10.1371/journal.pntd.0005768
- Fralick, J., Chadha-Mohanty, P., & Li, G. (2008). Phage Display and Its Application for the Detection and Therapeutic Intervention of Biological Threat Agents. In R. J. Kendall, S. M. Presley, G. P. Austin, & P. N. Smith (Eds.), *Advances in biological and chemical terrorism countermeasures* (pp. 179-202): CRC Press.
- Gorr, S. U., Flory, C. M., & Schumacher, R. J. (2019). In vivo activity and low toxicity of the second-generation antimicrobial peptide DGL13K. *PLoS One*, 14(5), e0216669. doi:10.1371/journal.pone.0216669
- Hahm, G., Glaser, J. J., & Elster, E. A. (2011). Biomarkers to predict wound healing: the future of complex war wound management. *Plast Reconstr Surg*, 127 Suppl 1, 21S-26S. doi:10.1097/PRS.0b013e3181fbc291
- Hwang, H. J., Ryu, M. Y., Park, C. Y., Ahn, J., Park, H. G., Choi, C., . . . Park, J. P. (2017). High sensitive and selective electrochemical biosensor: Label-free detection of human norovirus using affinity peptide as molecular binder. *Biosens Bioelectron*, 87, 164-170. doi:10.1016/j.bios.2016.08.031
- Kyte, J., & Doolittle, R. F. (1982). A simple method for displaying the hydropathic character of a protein. *Journal of Molecular Biology*, 157(1), 105-132. doi:[https://doi.org/10.1016/0022-2836\(82\)90515-0](https://doi.org/10.1016/0022-2836(82)90515-0)
- Marion, D. W., Curley, K. C., Schwab, K., & Hicks, R. R. (2011). Proceedings of the military mTBI Diagnostics Workshop, St. Pete Beach, August 2010. *J Neurotrauma*, 28(4), 517-526. doi:10.1089/neu.2010.1638
- Matplotlib. Retrieved from <https://matplotlib.org/>
- Mujahid, A., Afzal, A., & Dickert, F. L. (2019). An Overview of High Frequency Acoustic Sensors-QCMs, SAWs and FBARs-Chemical and Biochemical Applications. *Sensors (Basel)*, 19(20). doi:10.3390/s19204395
- Mukundan, H., Kumar S Fau - Price, D. N., Price Dn Fau - Ray, S. M., Ray Sm Fau - Lee, Y.-J., Lee Yj Fau - Min, S., Min S Fau - Eum, S., . . . Dickert, F. A.-O. Rapid detection of Mycobacterium tuberculosis biomarkers in a sandwich immunoassay format using a waveguide-based optical biosensor. (1873-281X (Electronic)).
- Python. Retrieved from <https://www.python.org>
- Tasoulis, T., & Isbister, G. K. (2017). A Review and Database of Snake Venom Proteomes. *Toxins (Basel)*, 9(9). doi:10.3390/toxins9090290
- Titus, J. K., Kay, M. K., Glaser, C. J. J., & Hwang, Y. Y. (2017). Application of phage display for the development of a novel inhibitor of PLA2 activity in Western cottonmouth venom. *J Venom Res*, 8, 19-24.
- Tkinter. (2020). TKinter Library. Retrieved from <https://docs.python.org/3/library/tkinter.html>
- White, S. (2011). Experimentally Determined Hydrophobicity Scales. Retrieved from https://blanco.biomol.uci.edu/hydrophobicity_scales.html

FIGURES

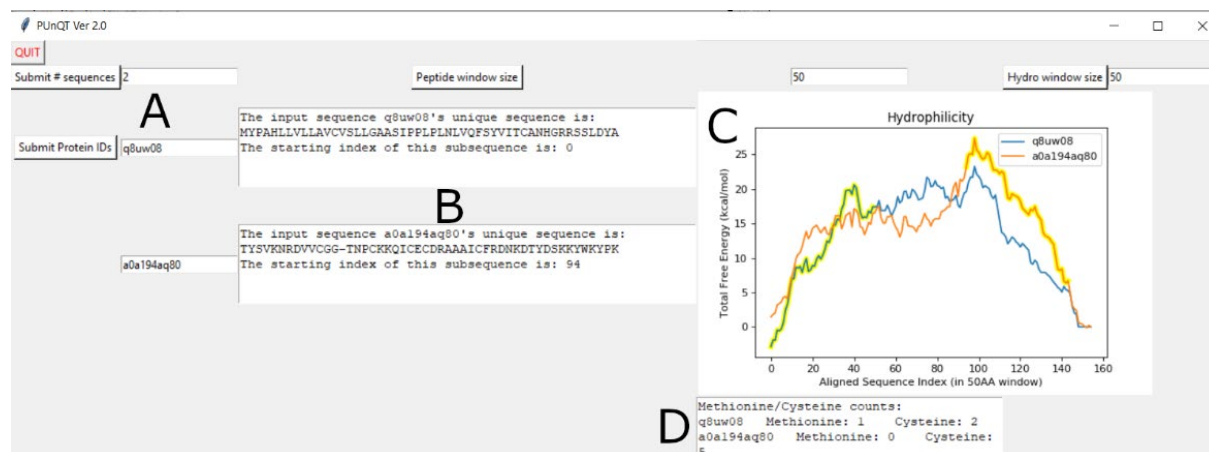


Figure 1. PUnQT graphical user interface and sample data. The graphical user interface of PUnQT is shown, with A) the number of aligned sequences requested/UniProt ID numbers, the B) calculated most unique subsequence for each aligned sequence, C) a hydrophilicity plot, and D) a methionine/cysteine count.

SOURCE CODE

```
# -*- coding: utf-8 -*-
"""
@author: Steve Moffett
"""
from tkinter import *
from Bio import ExPASy
import urllib
from urllib import request
from Bio import SeqIO
from Bio.Align.Applications import ClustalOmegaCommandline
import os
import matplotlib.pyplot as plt
import random
from PIL import ImageTk, Image
class App:
    def __init__(self, master):
    def fieldsallocator():
        """This function uses Tkinter's functionality to populate an
        initial GUI window."""
        self.num_seqs.configure(bg="white")
        self.numseqbutton.configure(bg="white")
        self.proteinID.configure(bg="yellow")
        seq_number = int(self.num_seqs.get())
        for i in range(1,seq_number+1):
            self.seqs.append(Entry(frame, bg='yellow'))
        rownum = 1
        for j in self.seqs:
            j.grid(row=rownum+1, column=1)
            rownum += 1
        def proteinIDsubmitter():
            """This function extends the Tkinter functionality to use the
            data
            in the fields of the UniProt ID numbers to directly query the
            website for sequence information"""
            seq_number = int(self.num_seqs.get())
            self.proteinID.configure(bg="white")
            for i in self.seqs:
                i.configure(bg='white')
            for i in self.seqs:
                seq0 =
                urllib.request.urlopen("http://www.uniprot.org/uniprot/"+i.get()+
                ".xml")
                seq_record = SeqIO.read(seq0, 'uniprot-xml')
                self.protdict[i.get()] = str(seq_record.seq)
                self.seq_record_list.append(seq_record)
```

```

self.seq_names_list.append(i.get())
"""This part of the function takes a list of seq records and
makes them into a fasta
file, then passes it to clustal omega for alignment. It
returns a list of
sequence records to the main function."""
namesString = ""
for i in self.seq_names_list:
namesString = i + "_" + namesString
handle = open(namesString+"resultfile.fasta","w")
for sequences in self.seq_record_list:
SeqIO.write(sequences,handle,"fasta")
handle.close()
#names in and out file for the clustal omega program
in_file = namesString+ "resultfile.fasta"
out_file = namesString+"alignedresultfile.fasta"
#this field should be changed to reflect the file path of the
#clustal omega executable location
clustalo_exe = r"C:\clustalomega\clustalo.exe"
assert os.path.isfile(clustalo_exe), "Clustal Omega
executable missing"
#performs sequence alignment
clustalo_cline = ClustalOmegaCommandline(clustalo_exe,
infile=in_file,\
outfile=out_file, verbose=True, auto=True)
aligned_object = clustalo_cline()
#writes sequence results to file
for seq_record in SeqIO.parse(out_file, "fasta"):
self.seq_list.append(str(seq_record.seq))
print("Sequences Aligned, results files available.")
self.window_size.configure(bg="yellow")
self.window_size_button.configure(bg="yellow")
def uniqueness():
self.window_size.configure(bg="white")
self.window_size_button.configure(bg="white")
self.hydro_window_size.configure(bg="yellow")
self.hydrophobicity_button.configure(bg="yellow")
window_size = int(self.window_size.get())
seq_lengths = []
for i in self.seq_list:
seq_lengths.append(int(len(i)))
if window_size > min(seq_lengths):
print("Too large peptide size. Please enter a shorter \
interval than the length of the query proteins: ")
else:
unique_ints, self.SA = uniqueness_marker(self.seq_list,
min(seq_lengths))

```

```

self.maxvalindex =
uniqueness_adder(unique_ints,windowsize, min(seq_lengths))
for keys in range(0,len(self.maxvalindex)):
print("The input sequence
"+self.seq_names_list[keys]+'s unique sequence is:")
print(self.SA[keys][self.maxvalindex[keys]:self.maxvalindex[keys
]+windows
ize])
print("The starting index of this subsequence is:
"+str(self.maxvalindex[keys])+"\n")
for i in range(0,len(self.maxvalindex)):
self.unique_subseq.append(Text(frame, width=60,
height=5))
rownum = 1
for j in range(0,len(self.maxvalindex)):
subseq_span =
str(self.SA[j][self.maxvalindex[j]:self.maxvalindex[j]+windowsiz
e])
start_ind = str(self.maxvalindex[j])
self.unique_subseq[j].grid(row=rownum+1, column=2)
self.unique_subseq[j].insert('1.0', "The input
sequence "+self.seq_names_list[j]+'s unique sequence is: \n")
self.unique_subseq[j].insert('2.0', subseq_span+"\n")
self.unique_subseq[j].insert('3.0', "The starting
index of this subsequence is: "+start_ind)
rownum += 1
def hydrophobicityfunc():
hydro, methionine, cysteine =
hydrophobicity(int(self.hydro_window_size.get()),self.SA)
plotter(hydro, self.seq_names_list,
int(self.hydro_window_size.get()), self.maxvalindex)
print("Methionine/Cysteine counts: \n")
self.metcys.grid(row=6, column=3, sticky=W)
self.metcys.insert('1.0',"Methionine/Cysteine counts: \n")
for i in range(0,len(self.seq_names_list)):
self.metcys.insert(str(float(i+2)),self.seq_names_list[i]
+ " Methionine: " + str(methionine[i]) + " Cysteine: "\
+str(cysteine[i])+"\n")
def uniqueness_marker(seq_recs, limiter):
'''This function will generate the uniqueness score for each
protein sequence
by comparing it with each letter in the same position of the
other sequences.
It will print each sequence and its general uniqueness score,
and it will
return a dictionary containing the number (first, second,
third...) of the

```

```

sequence you put in as a key, with a list of its uniqueness
ints as its
values. '''
uniquedict = ("Aviana Molecular Technologies," 2020)
SA = []
numseqs = len(seq_recs)
for i in seq_recs:
    SA.append(i)
    for i in range(0,numseqs):
        '''This loop goes through each sequence, its sub-loops
        comparing each of
        the other letters at that position (including its own).
        If it is unique,
        the uniqueness score for that sequence and in that
        position goes up. If
        it is the same, the uniqueness score does not change.'''
        uniuqeseq = []
        uniuqestr = ""
        for j in range(0,limiter):
            unique = 0
            for l in range(0,numseqs):
                if SA[i][j] == "-":
                    unique += 0
                elif SA[i][j] != SA[l][j]:
                    unique += 1
            else:
                unique += 0
            uniuqeseq += [unique]
            uniuqestr += str(unique)
        uniuqedict[i] = uniuqeseq
    return uniuqedict, SA
def uniqueness_adder(uniquedict, windowsize, limiter):
    """This function takes the dictionary containing the number
    of sequence
    (1,2,3...) as keys and their uniqueness score in a list, and
    the userspecified
    window size, and adds up the uniqueness score with
    a sliding
    window. Once it finds the first instance of the window with
    the highest
    value, it returns a new dictionary with the same keys but the
    values are
    the indices of the start of the max values. This data can be
    used with the
    stringArrays list to identify the character strings of the
    most unique
    part of the sequences."""

```

```

maxvalindex = {}
for key in range(0,len(uniquedic)):
    presentseq = uniquedic[key]
    scorelist = []
    startindex = 0
    wintotal = 0
    while startindex <= len(presentseq)-windowsize:
        for position in
            range(startindex,startindex+windowsize):
                wintotal += presentseq[position]
                scorelist += [wintotal]
        wintotal = 0
        startindex += 1
    maxvalindex[key] = scorelist.index(max(scorelist))
    return maxvalindex
def hydrophobicity(windowsize,stringArray):
    """This function both assigns a hydrophobicity value for each
    residue
    based on Stephen H White's hydrophobicity scale, and also
    calculates
    the hydrophobicity/philicity value along a window of userdefined
    size. It takes as windowsize integer and a list of strings
    containing
    sequence information, and returns a dictionary with the
    sliding
    window values as well as the number of methionine and
    cysteine
    residues in the sequence (important for peptide synthesis
    considerations)."""
    self.hydro_window_size.configure(bg="white")
    self.hydrophobicity_button.configure(bg="white")
    #hydroDict is based on Stephen H. White's hydrophobicity
    scale
    hydroDict = {'A': 0.33, 'R':1.00, 'N':0.43, 'D':0.50,
    'C':0.22, 'E':0.12, \
    'Q':0.19, 'G':1.14, 'H':-0.06, 'I':-0.81, 'L':-0.69, 'K':1.81,
    'M':-0.44, \
    'F':-0.58, 'P':-0.31, 'S':0.33, 'T':0.11, 'W':-0.24, 'Y':0.23,
    'V':-0.53, '.':0, '-':0}
    resultsDict = {}
    McountList = []
    CcountList = []
    #sums net hydrophobicity within window and also counts M and
    C residues
    for sequence in stringArray:
        McountList += [sequence.count('M')]
        CcountList += [sequence.count('C')]

```

```

wintotal = 0
startChar = 0
seqtotal = []
countdown = 1
while startChar <= len(sequence)-windowSize:
    for letter in range(startChar,startChar+windowSize):
        wintotal += hydroDict[sequence[letter]]
        seqtotal += [wintotal]
    wintotal = 0
    startChar += 1
while startChar <= len(sequence):
    for letter in range(startChar,startChar+windowSizecountdown):
        wintotal += hydroDict[sequence[letter]]
        seqtotal += [wintotal]
    wintotal = 0
    startChar += 1
    countdown += 1
resultsDict[sequence] = seqtotal
return resultsDict, McountList, CcountList
def plotter(hydrodict, names, hydrowindow, maxvalindex):
    """This function plots the hydrophobicity for presentation
    within
    the GUI frame, and saves a copy of it as a .png file for the
    user's
    records."""
    maxind = 0
    keyname = random.choice(list(hydrodict))
    x = range(len(hydrodict[keyname]))
    uni = 0
    mainlines = []
    namesString = ""
    for i in names:
        namesString = i+ "_" + namesString
    for key in hydrodict:
        y = hydrodict[key]
        mainline, = plt.plot(y, label=names[uni])
        mainlines.append(mainline)
        uni += 1
    plt.plot(x[maxvalindex[maxind]:maxvalindex[maxind]+hydrowindow],
    \
    y[maxvalindex[maxind]:maxvalindex[maxind]+hydrowindow],
    c='yellow',\
    lw=5, zorder=-1)
    maxind += 1
    plt.xlabel("Aligned Sequence Index (in
    "+str(hydrowindow)+"AA window)")
    plt.ylabel("Total Free Energy (kcal/mol)")

```

```

plt.title("Hydrophilicity")
plt.legend(handles=mainlines)
plt.savefig(namesString+'hydrophobicity_figure.png',dpi=80)
image = Image.open(namesString+'hydrophobicity_figure.png')
photo = ImageTk.PhotoImage(image)
label = Label(frame, image=photo)
label.image = photo
label.grid(row=2, column=3, columnspan=4, rowspan=4,
sticky=N+W)
return
"""The following sections of code initialize the frame for the
GUI
and then run the commands in the given functions above. It will
yield
a unique subsequence for each query sequence, as well as a
hydrophilicity
plot that assists in picking an external-facing part of the
protein. In
addition, it yields a count of methionines and cysteines in the
sequences,
which have bearing on the ease of peptide synthesis."""
#initializes GUI frame
frame = Frame(master)
frame.grid()
#Quit button
self.quitbutton = Button(
frame, text="QUIT", fg="red", command=frame.quit
)
self.quitbutton.grid(row=0, column=0, sticky=W)
#Label and field for Num of Seqs
self.num_seqs = Entry(frame, bg="yellow")
self.num_seqs.grid(row=1, column=1)
#button to generate fields for number of seqs
self.numseqbutton = Button(frame, text="Submit # sequences",
bg="yellow", command=fieldsallocator)
self.numseqbutton.grid(row=1, column=0)
self.seqs = []
#button to get protein codes --> seq records and name lists
self.proteinID = Button(frame, text="Submit UniProt IDs",
command=proteinIDsubmitter)
self.proteinID.grid(row=2, column=0)
self.protdict = {}
self.seq_list = []
self.seq_record_list = []
self.seq_names_list = []
#populates a window for the peptide window size button
#and calculates the uniqueness

```

```

self.window_size = Entry(frame)
self.window_size.grid(row=1, column=3)
self.window_size_button = Button(frame, text="Peptide window
size", command=uniqueness)
self.window_size_button.grid(row=1, column=2)
self.maxvalindex = {}
self.unique_subseq = []
#field to get hydrophobicity window size
self.hydro_window_size = Entry(frame)
self.hydro_window_size.grid(row=1, column=5)
self.hydrophobicity_button = Button(frame, text="Hydro window
size", command=hydrophobicityfunc)
self.hydrophobicity_button.grid(row=1, column=4)
self.metcys = Text(frame, width=40, height=4)
if __name__ == '__main__':
root = Tk()
root.title("PUnQT Ver 2.0")
app = App(root)
root.mainloop()
root.destroy()

```