Knot Again: Another Implementation of the Taylor Algorithm For Knot Detection

B. Jenner, S. Sastry, S. Shastry

March 13, 2020

1 Introduction

1.1 Proteins: An Overview

Proteins are dynamic molecular devices critical to a plethora of cellular functions, including cell-cell interactions, immuno-modulation, vesicular transport, and maintaining the structural integrity of cells [4]. The building blocks of proteins are only 20 amino acids but folded proteins may be as long as 27,000 amino acids or as short as 3 amino acids. This allows for simple proteins that have single domains as well as complex proteins with multitudes of domains and nuanced internal interactions. Regardless of their size, proteins are of great interest. Biotechnological and next-generation applications of proteins in the industry are being studied in great detail through protein vaccine engineering, sustainable bio-fuel production via enzymes, and food preservation through enzyme inhibitors [7].

1.2 Protein Structures

Before introducing the topic of study, it is necessary to explore what composes proteins. Proteins are the totality of their primary, secondary, tertiary, and in some cases, quaternary structures. The primary structure of a protein is the simplest aspect, defined by the order of amino acids linked together by peptide bonds. Though simple in concept, primary structures often determine a protein's downstream structural fates like folding and protein-protein interactions [11][3]. Flavors of 3D structures first begin to form in a protein's secondary structure: α helices, β pleated sheets, or a combination of the two depending on the relevant domain. The alpha helices resemble coiled springs and the beta sheets appear as folded pleats. Both alpha and beta structures are held secure via hydrogen bonding. Tertiary structures are more complex arrangements of a protein's secondary structure that result from disulfide bridges, van der waals forces, hydrogen bonds, ionic bonds, hydrophobic bonds, and R-group interactions. For many proteins, tertiary structures may be their last stage and is sufficient to define its biologically functional native state [11][3]. Lastly the quaternary structure, best thought of as an amalgamation of multiple polypeptide subunits forming a complete and working protein. Although tertiary structure determines protein folding and conformation, quaternary structure permits proteins to exhibit the necessary flexibility to harmonize protein-protein or protein-ligand interactions [12].

1.3 Protein Folding: Introducing Knots

During the formation of a protein's tertiary structure, the complex foldings sometimes give rise to knots, naturally embedding themselves deep within the protein's native state. The advent of next generation X-ray crystallography and high-throughput sequencing approaches has allowed scientists to represent proteins 3-dimensionally, sequence, and annotate a number of proteins; many of them recorded in the open-source GenBank database [10]. Additionally, the Protein Database (PDB) is another repository of protein structures, functions, sequences, and interactions [2]. In 2015, Jamroz et al. released KnotProt, a database of proteins with knots and slipknots [9]. Together, these efforts have aided our understanding of the complexity of proteins and provided insights about the relationship between structural, functional, and sequence dynamics in proteins [4]. Despite the advancements in protein structure determination, there is an ongoing pursuit to determine and characterize types and locations of knots in proteins as their biological functions are not yet thoroughly understood. Therefore, understanding the knotedness and mechanisms of protein folding is a worthwhile cause for predicting protein functionality and in vitro disease research.

1.4 Defining Knots

Understanding the folding nature of proteins can bring structural biologists closer to understanding their role in complex systems. Mathematically, knots are defined by a closed loop in three dimensional space which cannot be reduced or untangled to reveal a single loop [9]. It is important to acknowledge that our functional definition of a knot is based on the biologically existing protein backbone and overlaps very little with the mathematical definition of identifying knots. For the scope of this project, a knot in a polypeptide occurs when residues do not lie on a straight line between termini if the protein were to be pulled like ends of a string. When both termini are pulled, an active knot will not get disentangled. As a working definition for this algorithm, a knot in a protein can be quantified by the number of residues that need to be removed from each end to get rid of the knot.

1.5 The Humble Pursuit

Our algorithm aims to incorporate methods described in [6] to detect knots in an open chain protein [6][8]. The Taylor method was complemented with the Moller Trumbore algorithm for ray-triangle intersection, which is a robust computational tool for checking polypeptide intersections. We hypothesize that these methods will be sufficient for identifying knots in proteins as well as their locations on the polypeptide chain.

2 Methods

2.1 Program Structure

Our program, knotty.py, is a Python program consisting of two sub functions used to analyze protein structures— find() and visualize(). The first module find() contains our implementation of the Taylor knot-detection algorithm and is centered around the Python

"numpy" library [17], while the visualize() component is a 3D interactive tool that uses the Python library "Matplotlib" for visualizing the alpha carbon chain of your protein of interest [1]. Beginning with the "find" function, it's input is a text file in which each line denotes a 3D position vector for the alpha carbon of each amino acid residue. The intended output is both a similar text file describing a "smoothed" amino acid chain as well as a statement about knot detection at certain positions. When this function is executed, the text file is parsed and converted to a doubly linked list. This data structure was chosen due to its linear time complexity in forwards and backwards traversal, its instantaneous insertion/deletion operations, and its ability to contain node specific information. In our program, each node in our list is an amino acid and contains the 3D position vector of its central carbon atom along with its index in the chain and whether or not it was involved in the detection of a knot. Next in our algorithm, the list is traversed from positions 2 to n-1, leaving the termini untouched by the algorithm. At each index, the program attempts to "smooth" the chain by replacing the current node with an average of itself and its two neighbors (Figure 1, right). This maneuver is prevented, however, if the smoothing action crosses over another segment in the protein chain (Figure 1, right). This process is applied to the entire polypeptide repeatedly until the entire chain is colinear or the max number of iterations is reached. If more iterations are desired, the parameter can be updated.

Two polypeptides are displayed below. The left chain is unknotted whereas the right chain clearly contains a knot. Smoothing is achieved if and only if protein residues described by line segments preceding and following triangle planes created by Δ {i-1, i, i'} and Δ {i', i, i+1} are not intersected. If this condition holds, the position of the residue denoted by i is replaced with i', thus "smoothing" the polypeptide further. If this condition is not met, i is not replaced with i' and a knot is detected at the residue being evaluated. Repeated iterations of this procedure eventually approached colinearity.

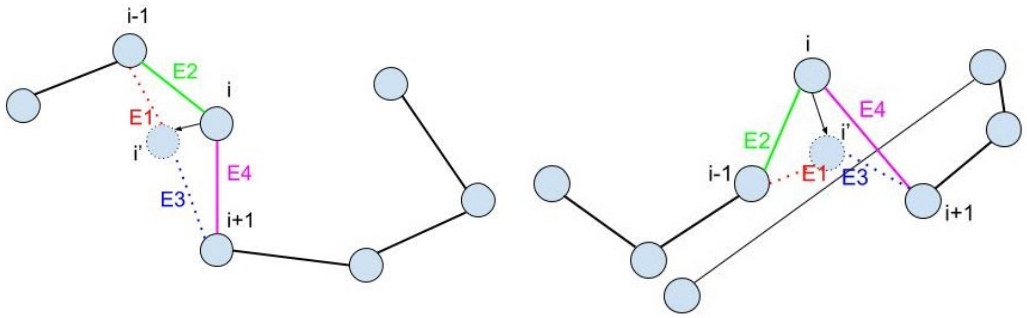

Figure 1: Visualizing Smoothing

2.2 Smoothing and Colinearity Approximations

As previously mentioned, the smoothing algorithm consists of taking the residues, which are represented as the coordinates of their a-carbon, and continually replacing each point by the average of itself and its two neighbours i' (Figure 1), with the existence of a line segment intersecting the triangle created by the three coordinate residues and the averaged

coordinate preventing this action. In theory, if this process is continually applied, colinearity will be approached among the residues between the two termini of the protein. However, this approach to smoothing is conservative and thus not time efficient. To increase the speed of this algorithm, a colinearity approximation function was implemented that checks the minimum distance separating a point and the line defined by its nearest neighbors. The parameter that defines this height cut off is called "epsilon " and is adjustable within the program. This approximation function accomplishes its task by first projecting the position vector of an amino acid onto the line described and solve for the height of the vector perpendicular to the point's projection by taking its magnitude. If the height calculated is below the specified threshold (epsilon), then the points are considered colinear and the central point is removed, creating a straight line between its two neighboring points. This approach reduces run time of this program by smoothing points quickly and decreasing the number of computations per loop in the algorithm.

2.3 Moller-Trumbore Algorithm

Another component of this algorithm is its ability to detect moves in the smoothing algorithm that would cross over upstream or downstream segments of the amino acid chain. This is essential because our current implementation of the smoothing algorithm has the potential to undo knots in the protein structure if these moves are not checked. In our program we used a set of criteria provided by Taylor's original paper on detecting knots that allowed us to smooth the protein chain and preserve existing knots. In Taylor's original paper, the method of evaluating the necessary criteria for smoothing is represented geometrically by two triangles formed by alpha-carbons {i-1, i, and i+1} and {i'}, where i is the current alpha carbon being considered and i' refers to the average of i - 1, i, i + 1.(figure- diagram in the original paper). The paper states that if a line segment, defined by the bond between any two alpha carbons upstream or downstream of i, intersects Δ{i - 1, i, i'} or Δ{i, i', i + 1}, smoothing is not allowed, preventing a potential knot from being smoothed. Our algorithm extensively uses this method of preventing illegal smoothing of the amino acid chain, yet through an implementation of the Moller-Trumbore Ray/Triangle Intersection algorithm (cite paper).

This method defines the intersection of a ray and a triangle as the point at which the ray equation (figure x1) and the equation for the barycentric coordinates of a point inside a triangle are equivalent (figure x2). This equation can be rearranged and converted into a matrix equation in which the solution is a three dimensional vector containing the coefficients "u" and "v" as well as the scalar "t". The coefficients stem from the barycentric coordinates equation, which define the point inside of a triangle as a linear combination of the vectors defining two edges of the triangle using the coefficients "u" and "v." The parameter "t," however, is from the ray equation and is used to define the length of the of the ray segment. In our case, however, if this value is larger than one, we know the ray segment does not reach the plane and does not intersect. Given the nature of matrix operations, one can calculate these parameters sequentially and thus prevent further calculation if a value is outside of a necessary range. For example, if either "u" or "v" is not between one and zero or their sum is greater than 1, we know that the point of intersection lies outside of the triangle and no more calculations are necessary to determine "t". Aside from ease of implementation, this is

$$R(t) = O + tD \qquad T(u,v) = (1-u-v)V_0 + uV_1 + vV_2,$$

Figure 2: Ray Equation (Left) and Barycentric Coordinates Equation (Right)

the primary advantage of the Moller-Trumbore Ray/Triangle Intersection algorithm. This method of systematically calculating each parameter and checking for incompatible values saves both time and memory, as calculations are minimized and computationally expensive plane equations do not need to be stored in memory as in other methods.

2.4 Protein Visualization

After the previous function is run and a file containing the coordinates of the smoothed protein is generated, one can now visualize this structure with the visualize() function in knotty.py. Similar to before, this function takes a text file with each line containing the 3D position vector of an alpha carbon in an amino acid chain, and converts it to an interactive 3D ball and stick diagram. The user interface allows one to zoom in and out on certain parts of the polypeptide, change its orientation, and even save PNG files of the image. Aside from this novelty, this extra functionality is intended for users to manually investigate smoothed protein chains for suspected knots or to observe if the smoothing algorithm applied was extensive enough to reveal a knot.

2.5 Usage & Testing

To test the functionality of knotty.py and obtain data on the performance and behavior of the program, we tested the find() function on multiple knotted and unknotted proteins of variable lengths and functional classes. Twelve proteins were taken from the examples given in class, recent publications, and the RCSB Protein Data Bank (Table 1) [5]. Tests were used for observing the relationship between protein sizes and runtime for different knot statuses. Additionally, the program was tested with different values for its epsilon parameters to observe its effects on runtime and false negative generation. All plots/graphs concerning the data generated during these tests were created in R. The visualize() function was used to demonstrate the effects of the smoothing algorithm for certain proteins. All unit tests were run on a 2017 Macbook Pro 13 with 8 GB's of RAM and a 2 core (4 threads) 2.3 GHz Intel CPU.

3 Results

3.1 Efficacy and Protein Reduction

Overall, our program succeeded in detecting the presence/absence of knots in 11 of the 12 proteins tested. The one instance in which the algorithm failed, however, is visualized in Figure 3, where a knot is clearly visible. The extent to which each polypeptide was reduced in overall size due to the colinear approximation function can be visualized in Table 1. On average, the algorithm reduced each protein in size by 47.9% while successfully predicting the presence/absence of knots using the find() function exclusively with 92% accuracy. This inability to identify knots with complete accuracy may have been due to the default epsilon value. Tighter thresholds were seen to identify knots much more reliably compared to looser thresholds, which may have accidentally produced a new knot in the process of "smoothing" a residue.

Table 1: Proteins used for unit tests and data collection

PDB Code	Knots / No Knots	Length (before)	Length (after)
4LRV	Knots	106	69
2LEN	Knots	231	173
1ZNC	Knots	262	99
2F69	Knots	257	106
1FUG	Knots	382	198
4KPP	Knots	394	312
5PTI	No Knots	58	25
1HDO	No Knots	207	82
1PVA	No Knots	218	108
1TIM	No Knots	247	120
2Y60	No Knots	330	148
4EPL	No Knots	551	274

3.2 Protein Length vs. Runtime

The results of Length versus Runtime test yielded statistically significant (p-value ¡ 0.05) positive correlations between the two variables with correlation coefficients of 0.886 and 0.962 for knotted and unknotted proteins, respectively (Figure 3). The maximum runtime observed was 3197.72 seconds (roughly 54 minutes) when given a 551 amino acid protein while the minimum runtime was 19.77 seconds for another unknotted protein containing only 56 amino acids. This was an expected results, as the algorithm operates with a rough time complexity of O((n2)k). Additionally, there did not seem to be a difference in runtime in proteins that were knotted versus those without.

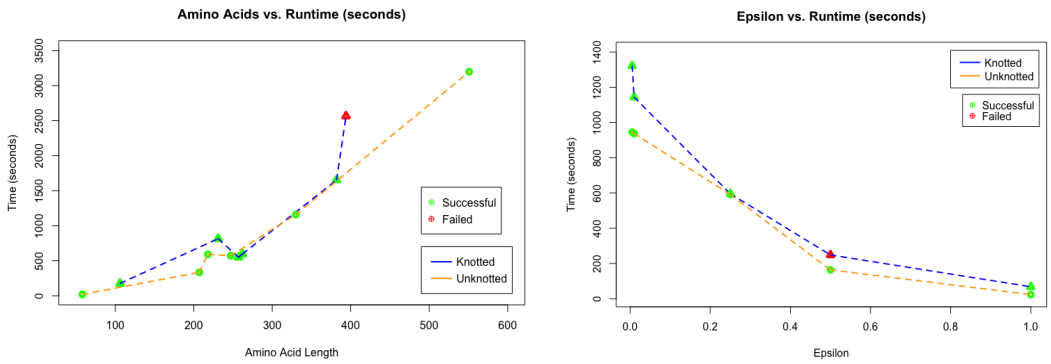

Figure 3: Length vs. Runtime (Left), Epsilon vs. Runtime (Right)

3.3 Epsilon vs. Runtime

While length of the amino acid was positively correlated with runtime, adjustments in the epsilon parameter revealed an opposite trend (Figure 3). As with length vs runtime, this analysis showed statistically significant (p-value ¡ 0.05) negative correlations between the two variables. These correlation coefficients were -0.918 for knotted proteins and -0.946 for proteins without knots. The maximum and minimum runtimes for these comparisons were 23.6 seconds from a knotted protein and 1321.72 seconds from a knotted protein which corresponded to the epsilon values of 0.005 and 1.0, respectively. Again, this was an expected result. The epsilon parameter had the potential to greatly reduce runtime by more generously approximating the collinearity of some points. Yet careless application of this algorithm can lead to false negative results, as evident in Figure 3. Curiously, a larger value for epsilon was quicker in terms of runtime and also generated a true positive result, leading us to believe there is no "one size fits all" for parameters and that knot detection in proteins is a difficult problem to generalize.

3.4 Visualization of Proteins

Results from the previous tests were visualized using the "visualization" function within knotty.py and allowed us to verify our results. As a result, we were able to detect an obvious knot in a particular protein that had failed our knot detection algorithm. Additionally, the knowledge this interactive visual provided allowed us to make modifications to our default parameters that allowed us to successfully detect the knot. The structure visualized in the pictures in Figure 4 are from this protein. In the first picture (left), we have the input coordinates file— a clear jumbled mess of a protein. In the right picture, we have the protein coordinate file after the original smoothing test with default parameters was applied. While this test failed in knot detection alone, upon looking at the graph, it was abundantly clear that this protein did in fact have a knot. Not only does this demonstrate the utility of the visualize function, but it also serves as a perfect example in which the default settings are not optimal for the protein of interest.

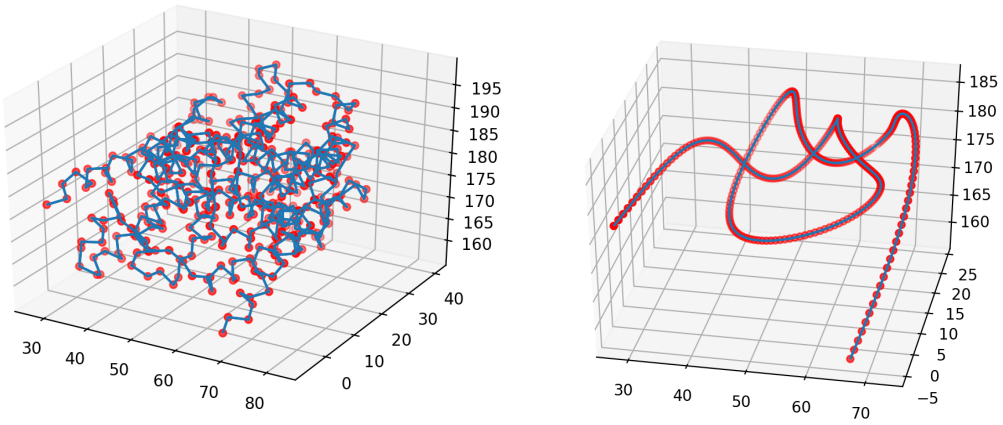

Figure 4: 4KPP: Original protein structure (Left), Computed protein structure (Right)

4 Discussion

One major limitation of this algorithm is the triangle checking function intersection(). This algorithm loops through "n" residues in the protein and runs intersection() twice for every residue, which in turn checks every line segment connecting two residues in the protein. This process is repeated "k" times where "k" corresponds to the maximum number of iterations, leading to a time complexity of O((n2)k). This may be unnoticeable when processing a smaller set of proteins, but it becomes a significant hindrance when larger proteins or datasets. One avenue that could be explored is the implementation of hybrid code. Python is a high level language and this brings with it a greater degree of abstraction compared to C or C++. Although we were unable to quantify the difference in processing speed in a pure Python program versus Python / C++, this could be a useful improvement when applying this algorithm to larger data sets. Another potential improvement could be removing the smoothing algorithm and just using the colinearity approximator instead while also limiting its moves using the same criteria.

Topological knots in tertiary protein structures are quite simple to identify, using the above method, as they consist of a single polypeptide chain. This method, however, disregards any R group interactions that may occur in the knotted regions and only address the spatial relationships of the alpha carbons. While ignoring these interactions allows us to identify locations of topological knots quickly, eliminating certain residues during the "smoothing" sections of the algorithm means that our analysis is purely topological and does not address any chemical interactions within the protein itself. The chemical compositions of proteins are as important as their physical structures and disregarding that element of a protein could lead to an incomplete understanding of how knots affect protein's functionalities. Implementing a method to address the chemical interactions within the protein, as the residues are smoothed, could allow us to identify the most prevalent amino acid residues in knotted regions of the protein and allow for a better understanding of how knots influence protein functionality.

5 Conclusion

What used to be an uncertainty, like the complex confirmations exhibited in a protein's native structure through folding nature and knotedness has progressively become more clear. Successfully building off of the novel Taylor and Moller Trumbore algorithms for determining protein knots via line segment intersections through triangular planes sheds light on a pursuit larger than this implementation. This algorithm has brought to the forefront gaps in structural biology that are actively being studied such as why proteins fold, mechanisms of achieving native structure, and the biological role of knots in proteins. Identifying protein knotedness is surely a necessary step in furthering this research. The premise of this method creates a dialogue in which scientists can engage with the true nature of proteins, full of approximations, assertions, and ultimately, a healthy sense of respect for the unknown.

References

[1] Tomas Möller and Ben Trumbore. "Fast, minimum storage ray-triangle intersection". In: *Journal of graphics tools* 2.1 (1997), pp. 21–28.

[2] Protein Data Bank. "Hm berman, j. westbrook, z. feng, g. gilliland, tn bhat, h. weissig, in shindyalov, pe bourne". In: *Nucleic Acids Res* 28 (2000), p. 235.

[3] Bruce Alberts et al. "The shape and structure of proteins". In: *Molecular Biology of the Cell. 4th edition*. Garland Science, 2002.

[4] Ardala Breda et al. "Protein structure, modelling and applications". In: *Bioinformatics in Tropical Disease Research: A Practical and Case-Study Approach [Internet]*. National Center for Biotechnology Information (US), 2007.

[5] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[6] William R Taylor. "Protein knots and fold complexity: some new twists". In: *Computational biology and chemistry* 31.3 (2007), pp. 151–162.

[7] Wangsa T Ismaya. "The Role of Protein Biochemistry in Biotechnology". In: (2011).

[8] Patrıcia FN Faısca. "Knotted proteins: A tangled tale of structural biology". In: *Computational and structural biotechnology journal* 13 (2015), pp. 459–468.

[9] Michal Jamroz et al. "KnotProt: a database of proteins with knots and slipknots". In: *Nucleic acids research* 43.D1 (2015), pp. D306–D314.

[10] Karen Clark et al. "GenBank". In: *Nucleic acids research* 44.D1 (2016), pp. D67–D72.

[11] Regina Bailey. "Proteins in the Cell". In: (2019).

[12] Renata Fioravanti Tarabini et al. "The importance of the quaternary structure to represent conformational ensembles of the major Mycobacterium tuberculosis drug target". In: *Scientific reports* 9.1 (2019), pp. 1–9.