Protein Structure Geometry

Madison Chamberlain, Zachary Babka, Lily Karim

Abstract

Understanding the geometry of a protein is essential for understanding its interactions and activity. We generated an algorithm which helps us understand this geometry by finding the volume and surface area of a protein structure defined as a union of all of its atoms, which are represented as spheres. In this program, the volume of a protein structure is calculated numerically using the Monte Carlo integration method. The program takes in a text file input containing the center coordinates and radius of each atom of the protein, as well as the number of randomly generated points the user wants to use for approximating the volume. We modified the program so that it could also calculate the surface area of the protein structure. We found that the accuracy for the volume and surface area calculations increased as the number of points used to generate these calculations increased. Since program runtime also has to be taken into account, we determined an optimal amount of points that should be used to generate volume and surface area of the protein.

1. Introduction

A protein is a sequence of amino acids that interact with adjacent and distant amino acids to form overlapping structures. Because these structures are highly complex and very specific, determining its characteristics such as volume and surface area provide insights to the activity, solvation, and interactions of the protein with other molecules and its environment [1][2][3]. Mathematical modeling is essential for visualizing and assessing properties of the protein due to the number of amino acids and overlaps each different protein contains[1][3].

The protein can be modeled as adjoining spheres which represent atoms with varying locations and sizes. These spheres are represented in 3 dimensions with the x,y, and z coordinate of the center point and the radius in angstroms [1]. Due to the sheer variety of proteins and their amino acid sequence and structure, standardizing and optimizing the protein volume calculation is essential. Calculating the volume of a protein requires many computational, temporal, and mathematical considerations and as a result, many methods have been developed for optimizing the accuracy and efficiency of this calculation. Using this sphere modeling method, the volume of the entire protein can be calculated using either an analytical or numerical approach.

One analytical approach of this problem is inclusion-exclusion, which solves for the exact volume of the protein by calculating the total volume of the unions between the spheres and subtracting this from the sum of the total volume of the spheres[1][4]. The volume and surface area of the overlapping spheres can be solved with an alternating sum of volumes and surface areas of the common intersections of

the spheres. This method, as is true of most analytical methods, is computationally not feasible due to the exponential run time of calculating the intersection of N number of spheres and deleting redundant overlaps as more intersections are analyzed[1][4].

The direct inclusion-exclusion approach uses alpha shapes or complexes: the part of the Voronoi regions of the atoms that does not have redundant atom intersections[1] [4]. The alpha complexes represent multiple intersections without the need to subtract redundancies, allowing for the calculation of the volume of unions without infinite alternating sums [3][4]. This method, however, is still mathematically complex and challenging to accomplish.

Of the numerical approaches, the most feasible for computing this problem is the Monte Carlo Integration Method. This method estimates the volume of the protein by enclosing the overlapping spheres in a 3 dimensional rectangular prism and calculating the ratio of randomly generated points that fall inside the volume of the protein to those that land within the cube only. This ratio is used to approximate the volume of the protein as a percentage of the total volume of the cube. Variables and programming can be adjusted to improve the accuracy and speed of this calculation for the protein [3].

Monte Carlo integration has been applied more specifically to proteins to search for pockets in the structure of a protein in which ligands can access and potentially bind. In this implementation of the Monte Carlo method, the points used to assess the pockets of the protein are generated via a uniform grid[5]. These specific applications of the Monte Carlo method show how numerical computing methods provide useful, important, and reliable biological information.

In the methods used for solving for the geometry of a protein, special emphasis is placed on finding its exposed surface area as an indication of its solvency and interaction kinetics [1][2][3][4][5]. Surface area calculations are often approached with computationally costly geometric methods that take long times to compute and use highly complicated calculations to determine overlap [1][6].

Based on previous studies, we have selected the numerical Monte Carlo integration method to solve for the volume of a protein and a modified concept based on the Monte Carlo integration method to solve for the surface area of a protein. Both of these characteristics are essential to understanding important activities and features of the protein. This numerical approach is best suited to the parameters of our program because it provides simpler calculations and faster, less costly run times while still offering accurate approximations.

2. Methods

The Monte Carlo method of integration is different from classical integration in that it numerically integrated random points to approximate a final value. We used a form of Monte Carlo integration to solve for the value of both the volume and surface area of a protein structure defined by a union of spheres. The algorithms we

developed to determine the surface area and volume of the protein structure are inherently different, but they both are based in Monte Carlo technique.

2.1 Finding The Volume of a Protein:

To find the volume of the protein structure, we developed an algorithm which generates random points within a cube containing the protein, determines a ratio of how many of these points were in the protein to how many points were generated, and then multiplies that ratio by the volume of the cube.

The way this algorithm works specifically, is by taking as input a .txt file and a number of points, "N", the user would like to randomly generate to determine the Monte Carlo ratio. The .txt file must contain four columns containing an x, y, and z coordinate, and a radius in that order. Each row of the file represents one sphere, and if the input file begins with any other information, these lines must be removed from the file.

Our program begins by reading in this .txt file and converting it to a list of lists. Each internal list contains the x, y, z and radius of a single sphere so that we can easily iterate through this information. After we have read in the data, we begin to calculate the volume of the structure with Monte Carlo integration. First, we generate a rectangular prism that encompasses the entire protein structure. This prism is generated by finding the minimum and maximum values of x, y, and z between all spheres and subtracting the maximum radius value from the minimums and adding the maximum radius value to the maximums. This creates a prism that includes the volume of every single sphere.

Next, we randomly generate a list of x, y, and z points that are within the bounds of this rectangle. This list of points will be of length "N". Next we must check if each of these randomly generated points is within the protein structure. We do so by looking at each sphere within the protein structure; if the distance between the random point and the center of the sphere is less than or equal to the radius, the point is within this sphere. If we find that the point is within the sphere, we must keep track of this information. We do this by adding "1" to the number of random points that have been determined to lay within the protein structure. We repeat this process of checking whether a point is within each sphere for each randomly generated point.

Next we find the ratio of points that fell within the protein structure by dividing the number of randomly generated points that fell within the structure by the total number of points, or N. To calculate an estimate for the volume, we must multiply this ratio by the volume of the cube containing the protein structure. Due to the fact that this is an approximation, and not an analytical solution to finding the volume, we must add and subtract the standard error to our calculation of the volume to give a range for the estimate of the volume. We find the standard error by taking the ratio we calculated and multiplying it by one minus the ratio. Next we divide the number we calculated by N and then take the square root of that number to get the ratio of the

standard error of the volume. To get the actual standard error of the volume, we multiply this new ratio we calculated by the volume of the cube. When we add and subtract the standard error from the value of the volume we calculated, we are able to provide the user with an accurate range for the volume of their protein structure.

Although using the monte carlo method to calculate the volume of protein structure is less costly than the analytical approach would be, it is still time consuming. Since we have to check if *each* point is *any* of the spheres, the Big-O time complexity will be O(m*n) where m is the number of spheres in the protein structure and n is the number of points we use to generate a volume.

2.2 Finding the Surface Area of a Protein :

For calculating the surface area of the 3-dimensional protein structure, a different approach was needed. Randomly assigning points in the volume containing the protein and testing if any of them landed on the surface area of the structure would require N to be on the order of billions to generate a significant number of points that fell exactly on the surface of the protein. Because the surface area is a much smaller set of points compared to the volume of the protein, our previous approach for volume is not practical for either run time or accuracy when computing surface area. The alternative approach we used to overcome this efficiency and accuracy problem was to generate lists of random points on the surface of each sphere and test to see how many of these are actually on the surface of the protein as a whole. Then by figuring out the ratio of the randomly generated points that are actually on the surface of the union of spheres, we can figure out the exposed surface area for an individual sphere in the protein by multiplying the ratio of exposed points to total points by the surface area of that sphere. We can then sum the individual contributions from each sphere in the union of balls to find the total surface area of the protein. As the number of random points on each sphere approaches infinity, the estimated value of the surface area approaches the actual value.

To randomly generate a given number of points per sphere, we created a function called list_of_points_on_sphere that takes input coordinates for the center of the sphere, it's radius, and a number of points, "n". To generate random points on the surface of a sphere, we randomly generate values for the spherical coordinates theta (from 0 to 2 pi) and phi (0 to pi), and then use those values with the radius to convert back to a list of points in the same cartesian coordinate system as the input file for the program. With this function defined we can start our outermost loop, that iterates through every sphere in the union of balls. For each sphere it initializes a counter called "exposed points", that will eventually define the number of our randomly generated points for that sphere that are on the actual surface of the protein. The next loop iterates through each randomly generated point for that sphere. Within that loop, is another loop that iterates through all the other spheres to determine whether the point of interest lies

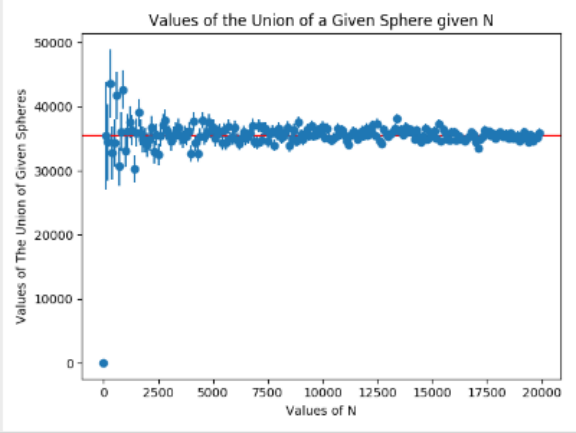
within any of the other spheres by comparing the distance of the point from the center of the sphere to that sphere's radius. The total number of exposed points is then counted and the contribution of surface area from this sphere is added to the total surface area of the protein.

Because this algorithm involves iterating through three nested loops, it is more time costly than the Monte Carlo method for estimating the volume of the union of balls. For the given protein with m atoms, we can estimate the Big-O time complexity as $O(m(m-1) * n^2)$ where n is the number of points per sphere being used to approximate the surface area of the protein. The m * (m-1) term is based on the fact that the main function has to iterate through each sphere to get its contribution to the surface area by comparing it to every other sphere. The n^2 term comes from the fact that each time the function is called, it calls the sub function runs n times to generate a list of length n. Then within the main function, the code iterates through each of the points in that list n times, giving this section n^2 complexity.

3. Results

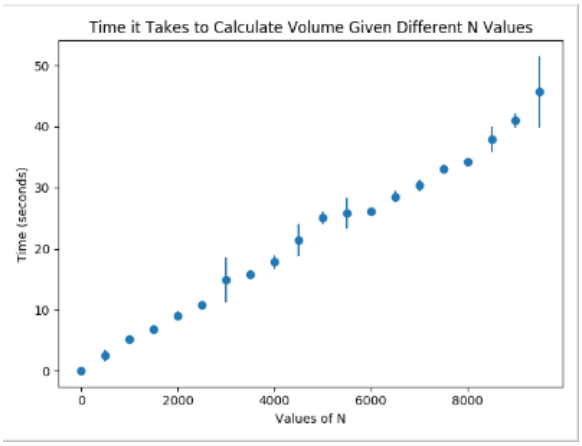
3.1 Finding the Volume of a Protein:

When we ran our volume-calculation-algorithm for a test case using a file that gave the information about a protein made up of 2,775 spheres, our function returned the following volumes given values of N from 1 to 20,000.



As seen in the graph, the standard deviation for the volume of the protein structure approaches zero as N gets larger. This means that in order to achieve a highly accurate volume interval, a large N should be used to calculate the volume.

We also graphed the time it took to calculate a volume given different values of N to attempt to determine where the sweet spot is where time is minimized and accuracy is maximized.

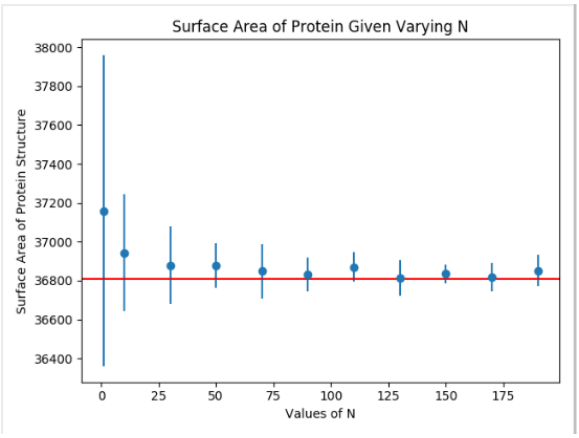


As seen above, the graph of time complexity looks relatively linear. This means that the time it takes to calculate a volume for the protein structure is directly proportional to the size of N. To minimize time and maximize accuracy, we looked at N = 3,000

and N = 20,000 to determine how much accuracy improves as a function of time, and to judge the best N to use for fast results. We found that when N = 3,000, the volume is found to be 35100.73 with a standard deviation of 1543.73, and this takes 14.34 seconds. When N = 20,000 the volume is found to be 35196.24 with a standard deviation of 598.55, and this calculation takes 97 seconds. This is roughly a 700% increase in time to generate a 300% increase in accuracy. We determined that 3,000 points is a good starting point for generating an accurate volume for the protein structure considering it's accuracy given its time.

3.2 Finding the Surface Area of a Protein:

When we ran our surface-area-calculation algorithm for the same test case, the protein made up of 2,775 spheres, our function returned the following surface areas given values of N from 1 to 190. Keep in mind that the number of points *total* is equal to N times the number of spheres.



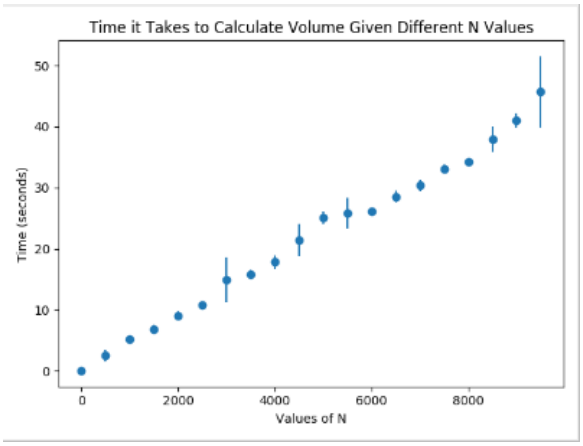
As you can see in the graph above, as N approaches infinity, the standard deviation of the surface area calculation approaches zero. This means that in order to achieve a highly accurate surface area interval, a large N should be used to calculate the surface area. Once again, it takes a long time to run this algorithm, and we wanted to determine the best N to use given accuracy and time. This task was much easier for surface area than it was for volume, since the standard deviation was statistically the same from N = 110 to N = 190. When N = 110, the surface area was calculated to be 36868.79, with a standard deviation of 75.92. When N = 190, the surface area was calculated to be 36852.17, with a standard deviation of 78.35. These results are nearly identical despite the fact that it takes around 49 minutes and 9 seconds to run N = 190, and around 26 minutes and 45 seconds to run N = 110; nearly half the amount of time it took to run N = 190. Thus, we have determined that setting N to 110 is the highest we need to set N to an accurate surface area calculation.

4. Discussion:

Inorder to make our algorithm run as efficiently as possible, while also still using the same basic procedure used, we examined where our code was costly. We first observed the fact that everytime we checked the distance between a randomly generated point and the center of a sphere, we compared the radius with the distance, which required us to use the square root function. This is a costly function considering the fact that it can easily be removed. To remove the square root, we simply compared the distance squared with the radius squared to improve runtime. Both of our functions needed to check if a point

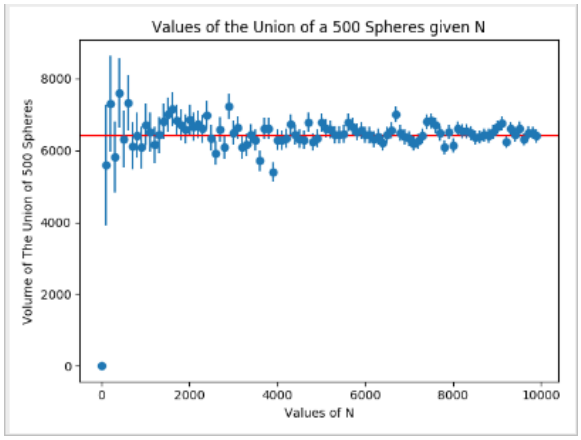
was overlapped by another sphere. Originally, our code checked for overlap by all other spheres even after a point was found to be overlapped. We changed the functions to include a break statement after it was determined that a random point is within another sphere. This reduces the run time for this loop from m to an average of 1/2 m. We approximate this from an average of the best case and worst case. If all points are overlapped, the break statement entirely eliminates the loop after the first iteration. However if none of the spheres overlap, this break statement is never run, and the loop's run time is still m.

Although both of these fixes only change the coefficients of the run time which will not actually alter the big-O, we know that this change will optimize the runtime given the functions we are using. The new big-O for the volume function is now O(½(n*m)), and for the surface area it becomes O(m(m-1) * (½n)(n)), where m is the number of spheres, and n is the number of random points we generate. We graphed the new time complexity of the volume function given N for comparison as seen below.



In the graph, you can see that the function is still linear, and it doesn't appear that we used a high enough N to really see the effect of the changes we made. However, the points in the line appear to have generally smaller standard deviations than the original graph, and we are confident that our algorithm runs as efficiently as possible considering the functions we included in it.

In order to optimize the usability of our algorithm, we sought to determine whether the optimal N changes given the number of spheres making up a protein. We did this by examining the volume output for the first 500 spheres from the original 2,775 sphere protein structure we have been looking at. The following graph shows how volume changes as N increases when there are 500, rather than 2,775 spheres.



As seen in our graph, the standard deviation for 500 spheres also started to get closer to zero at around N = 3,000 as well. We suspect that this trend may be due to the fact that the 500 and 2,775 spheres are from the same data set, which means the points are from within the same x, y, and z range. We suspect that if we had used a different dataset for the 500 points that we were

testing, which were guaranteed to all be heavily overlapping as a real protein structure would be, the standard deviation would approach zero at around the number of spheres that made up the protein structure. We feel that this question poses a good opportunity to expand upon our research in the future.

In terms of our surface area algorithm, we do not suspect that the optimal value for N should change given a protein made up of a different number of spheres. This is because the N entered by the user is actually the number of points per sphere tested, whereas for volume, N is the number of points total. This means that the optimal N to use when finding surface area will always be 110, since the total N is already dependent on the number of spheres in the protein structure.

5. Conclusion:

We were successful in generating an algorithm that takes in a .txt file containing center coordinates and a radius for each atom that makes up a protein structure, and returns an estimate for the volume and surface area of the protein structure. We were able to optimize the runtime for our algorithm by adding a break statement, and reducing our use of complex mathematical functions where ever possible. We found that to generate an accurate value for the volume of the protein, the user should input that they would like to generate the same amount of points as the number of spheres in the protein structure. We found that to generate an accurate surface area, the user should input that they would like to generate around 110 random points per sphere.

While our programs returns fairly accurate answers, within a percent error, it requires highly specific input for calculation which may not be practical for most applications. To improve this, this program could be combined with other programs that predict the 3-D structure of proteins from amino acid sequences, then map that to a model of a union of spheres representing each atom. This way, the program could return accurate surface area predictions from either DNA or amino acid sequencing. This would provide a much more practical application of our algorithm since it requires a less specific input and gives the program more versatility.

6. Bibliography:

1. Li J, Mach P, Koehl P. Measuring the shapes of macromolecules - and why it matters. *Comput Struct Biotechnol J*. 2013;8:e201309001. Published 2013 Dec 9. doi:10.5936/csbj.201309001

2. Mach, Paul, and Koehl, Patrice. "Geometric measures of large biomolecules: surface, volume, and pockets." Journal of computational chemistry 32.14 (2011): 3023-3038.

3. Till, M.S., Ullmann, G.M. McVol - A program for calculating protein volumes and identifying cavities by a Monte Carlo algorithm. *J Mol Model* 16, 419–429 (2010). https://doi.org/10.1007/s00894-009-0541-y

4. Liang, Jie, et al. "Analytical shape computation of macromolecules: I. Molecular area and volume through alpha shape." *Proteins: Structure, Function, and Bioinformatics* 33.1 (1998): 1-17.

5. Hendlich, Manfred, Friedrich Rippmann, and Gerhard Barnickel. "LIGSITE: automatic and efficient detection of potential small molecule-binding sites in proteins." *Journal of Molecular Graphics and Modelling* 15.6 (1997): 359-363.

6. Wodak, Shoshana J., and Joel Janin. “Analytical Approximation to the Accessible Surface Area of Proteins.” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 77, no. 4, 1980, pp. 1736–1740. *JSTOR*, www.jstor.org/stable/8425. Accessed 10 Mar. 2020.