Computing a New Family of Shape Descriptors for Protein Structures

Peter Røgen*,† and Robert Sinclair‡

Department of Mathematics, Technical University of Denmark, DK-2800 Kgs Lyngby, Denmark, and Department of Mathematics & Statistics, The University of Melbourne, Parkville, VIC 3052, Australia

Received May 14, 2003

The large-scale 3D structure of a protein can be represented by the polygonal curve through the carbon α atoms of the protein backbone. We introduce an algorithm for computing the average number of times that a given configuration of crossings on such polygonal curves is seen, the average being taken over all directions in space. Hereby, we introduce a new family of global geometric measures of protein structures, which we compare with the so-called generalized Gauss integrals.

INTRODUCTION

Some biological questions concerning closed double-stranded DNA[1,2] and the topologically active enzymes topoisomers[3] are naturally formulated and understood in terms of knot theory. Whereas DNA constitutes a library, the proteins, synthesized from the information held in DNA and correctly folded, actually perform the processes of life. In contrast to DNA, proteins are open curves and are thus not a priori amenable to knot theory.

The shape of a protein is known to be very important for its function. Throughout this paper we consider globular proteins in native states only. To be able to talk about protein shapes, structural classification systems have been developed, with differing degrees of automation. For example, SCOP[4] is a classification based upon expert human judgment, whereas CATH[5] has a high degree of automation. Common for the classifications of protein structures is that if two proteins are of similar structure (when measured by some similarity measure), then they are said to have the same topology.

Mathematically speaking, that is, when using the word “topology” as introduced by Johann Benedict Listing in 1847, all proteins have the same topology. Considering shoelaces, however, we all properly agree on some extremes. For example, we all agree that there is a difference between a tied and an untied shoelace. In the following, we will use the word “topology” as commonly done in the context of protein structures.

Proteins generally assume rather sharply defined structures. However, for some topological structures there are proteins with shapes between the extreme structures. These are referred to as structures in “twilight zones”.[6] Thus, in parts of a protein structure space containing twilight zones, the preferred structures are not of a discrete or topological nature. Furthermore, when more and more protein structures are found, new twilight zones are to be expected. Hence, future classification systems cannot be of an entirely discrete nature.

Proteins with known structures have now been divided into hundreds of topological classes, and the number of such classes is expected to grow to from 1000 [7] to 2000.[8] The number of topological classes and the speed at which new protein structures are found make it desirable to have a rigorous mathematical description of the topology of a protein structure, to automate protein classification (and thus make classification reproducible).

The main idea in the papers of refs 9 and 10 is to use a set of global geometric measures that each pick up a specific piece of topological information of a single given protein structure, rather than constructing similarity measures for given pairs of protein structures. Each protein structure is then characterized or represented by the values of these measures, that is, as a point in some Euclidean vector space of dimension n—equal to the number of measures. The value of each measure need only be calculated once for each protein structure. Given such measures, structural comparison is simply given by a metric on $\mathbf{R}^n$. Hence all-against-all structural comparison becomes very fast, and the computation time of the measures of any one protein becomes less important.

In the refs 9 and 10, so-called generalized Gauss integrals are used as structural measures. These measures are shown to be very powerful in the context of protein classification.[10] A simple automated protein structure classification system based on the generalized Gauss integrals reproduces the classification of connected CATH (version 2.4) domains with 96% success.

However successful, the generalized Gauss integrals can be argued to suffer from not having a simple geometric interpretation (except for the case of almost planar curves on which each generalized Gauss integral counts how many times a specific configuration of crossings occurs). This has motivated us to consider computing the average number of times each configuration of crossings occurs, taking the average over all directions in space, for which an algorithm and its implementation are the issues of this paper. Further-

* Corresponding author fax: (+45) 4588-1399; e-mail: Peter.Roegen@mat.dtu.dk.
† Technical Unversity of Denmark.
‡ The University of Melbourne.

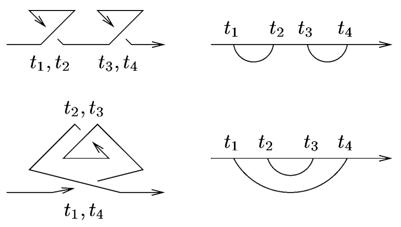

Figure 1. Top left: Two consecutive loops. Both crossings are left-handed. Top right: Configuration, denoted (1, 2)(3, 4), of the parameter values corresponding to two consecutive loops. Bottom left: One loop inside another loop. The inner crossing is left-handed and the outer crossing is right-handed. Bottom right: Configuration, denoted (1, 4)(2, 3), of the parameter values corresponding to a loop inside another loop.

more, ref 9 gives a heuristic geometric, statistical explanation of why each of the Gauss integrals considered in ref 9 is expected to be strongly correlated with the average occurrences of a specific crossing pattern. Here, we can verify that this is true for protein domains of lengths up to 188 residues for patterns of two crossings but in general not for patterns of three crossings.

AVERAGE OCCURRENCES OF CROSSING CONFIGURATIONS

Given a protein structure, we first reduce it to the polygonal curve connecting the carbon α atoms in the direction of increasing residue number, to represent the large-scale structure of the protein backbone. In the following, we are referring to this polygonal curve when we speak of the shape of a protein. Browsing through the protein classifications SCOP and CATH, it is clear that two protein structures are usually defined to belong to the same class if a loop of the one structure is longer or deformed relative to the corresponding loop of the other structure. In the C*A*TH classification system an *A*rchitecture is given by the relative positions of the secondary structure elements. An architecture of CA*T*H is then subdivided into several *T*opologies according to the threading of these secondary structure elements. In general, it seems that human observers allow deformations of the planar projections of the structures that do not involve switching of crossings or more general rethreading of the structures, which, in the case of a closed curve, can change its knot type.

On the basis of these observations, we believe that a good objective observer of a protein structure will note how many times any crossing pattern occurs from each direction in space and report the average over all directions of these observations. Figure 1 illustrates two of the three crossing patterns with two crossings. The third is the (1, 3)(2, 4) pattern, the average occurrences of which we denote $\langle(1, 3)(2, 4)\rangle$.

To count the number of patterns with m crossings, denoted by N_m, consider the first arc, that is the one starting at t_1; see Figure 1. The remaining $m - 1$ arcs can form N_{m-1} distinct patterns. Each of these N_{m-1} patterns connect $2(m - 1)$ points on the line. The first arc may end in each of the $2(m - 1) + 1$ intervals into which these $2(m - 1)$ points divide the line. Hence, $N_m = (2m - 1)N_{m-1}$. Now, since $N_1 = 1$, we get $N_2 = 3 \times 1 = 3$, $N_3 = 5 \times 3 \times 1 = 15$, and in general $(2m - 1) \times ... \times 7 \times 5 \times 3$ patterns with m crossings.

We have implemented an algorithm for calculating the average occurrences of patterns with one, two, and three crossings averaged over all directions. These correspond to the generalized Gauss integrals of order one to three considered in ref 10.

One can count the number of single crossings in both a signed and an unsigned manner. Signed counting uses the right-hand rule; i.e., a right-handed crossing is counted as one positive occurrence and a left-handed crossing is counted as a negative occurrence. For closed curves, this signed sum is known as writhe[1] or self-induction and may be calculated by the well-known integral found by Gauss. The unsigned sum is usually denoted the average crossing number.

For double crossing patterns there are four ways of counting occurrences, namely, always positive, using the sign of the first or the second crossing or using the product of the signs of the two crossings.

Triple crossing patterns have eight different sign conventions, but we only use the product of the three signs to keep the number of global geometric measures at a reasonable level (and the results of the section Separation of Protein Folds confirm that we have not chosen too few). Having made these choices, the measures considered here correspond to the generalized Gauss integrals considered in ref 10.

Levitt[11] used the writhe to distinguish different chain threadings. Banchoff[12] was the first to reduce the writhe integral of polygonal curves. The average crossing number is well-studied as a protein structure descriptor; see, e.g., the works of Arteca, such as ref 13. The average crossing number is basically given by the size of the protein[14] and is thus poor as a shape descriptor. Røgen and Bohr[9] found that the average crossing number is the only Gauss integral that can be predicted by the α, β, and coil contents of a protein. Further remarks on the average crossing number of protein structures are given in ref 9.

A right-handed helix remains right-handed if its direction of traversion is reversed. A structural measure that does not depend on the direction of traversion is referred to as symmetric in ref 9. In contrast to this, the four pairs of crossing patterns of three crossings, (1, 2)(3, 5)(4, 6) and (1, 3)(2, 4)(5, 6), (1, 2)(3, 6)(4, 5) and (1, 4)(2, 3)(5, 6), (1, 3)(2, 6)(4, 5) and (1, 5)(2, 3)(4, 6), and finally (1, 4)(2, 6)(3, 5) and (1, 5)(2, 4)(3, 6), gets pairwise-interchanged under reversion of the direction of traversion. All other patterns of one, two, and three crossings are symmetric. The mixed sign conventions on patterns of two crossings are introduced in order to get more structural measures that can discriminate between a structure and its reversion.

THE ALGORITHM

Take an open curve consisting of N line segments, specified by $N + 1$ points, and label these such that the endpoints of the line segment L_i $(1 \leq i \leq N)$ are P_i and P_{i+1}. Consider an orthogonal projection of three-space into a plane. If the two line segments L_i and L_j are seen to cross under this projection, then connecting the point on L_i that corresponds to the crossing point of the projection to the corresponding point on L_j gives a vector in three-space that

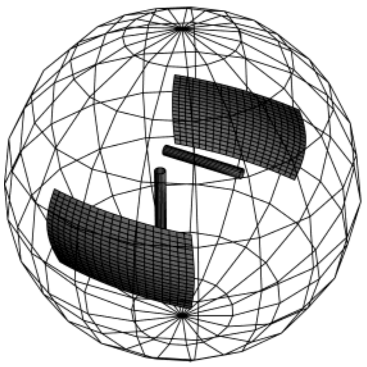

Figure 2. Normals of planes in which the two projected line segments are seen to cross, shown on the sphere.

is normal to the plane of projection. By normalizing this vector, we get one of the two unit normal vectors of this plane. The set $A_{i,j}$ of normals to planes in which the projection of the two line pieces L_i and L_j are seen to cross is thus given by $\pm(p_j - p_i)/(|| p_j - p_i||)$, where p_i lies on L_i and p_j lies on L_j. This is shown in Figure 2.

A piece of the boundary of the set $A_{i,j}$ is given by plus or minus $(p_j - P_i)/(|| p_j - P_i||)$, where P_i is the starting point of L_i and p_j lies on L_j. Hence, the boundary of $A_{i,j}$ consists of pieces of great circles on the unit two-sphere. If we define e by $e(v) = v/|v|$ for $v \in \mathbb{R}^3$, then the set $A_{i,j}$ is the union of two convex spherical quadrilaterals, one with corners at $e(P_i - P_j)$, $e(P_{i+1} - P_j)$, $e(P_{i+1} - P_{j+1})$, and $e(P_i - P_{j+1})$ and the other with corners at $- e(P_i - P_j)$, $- e(P_{i+1} - P_j)$, $- e(P_{i+1} - P_{j+1})$, and $- e(P_i - P_{j+1})$.

The most computationally intensive part of our computation of measures is the calculation of the areas of the intersections

$$A_{i,j} \cap A_{k,l} \cap A_{m,n}$$

where $i \leq j + 2$, $k \leq l + 2$, $m \leq n + 2$, and $iN + j < kN + l < mN + n$. This intersection contains all normals to planes in which L_i is seen to cross L_j, L_k is seen to cross L_l, and L_m is seen to cross L_n. The areas of these intersections are summed in various ways, often involving changing signs. In the following we will concentrate on the areas themselves.

One could in theory use Monte Carlo methods to compute these areas, and we have indeed used Monte Carlo integration extensively in the testing of our algorithm, but we have every reason to expect that the convergence of Monte Carlo methods can only be erratic at best in the present context. The reason is that we are essentially computing the integral of indicator functions on a sphere. These indicator functions are both discontinuous and typically of small measure. An extremely large number of points would be required to hit the smaller intersections even once.

In ref 13 the areas from which any given number of crossings is seen are calculated using Monte Carlo methods. The average crossing number is known to change at the order of 1% between similar protein structures.[9] Hence, one would like to calculate the writhe and the other measures with an accuracy that seems hard to obtain by the methods of ref 13. We choose instead to eliminate the uncertainty coming from Monte Carlo methods and implement a theoretically exact algorithm for calculating the measures.

Computational Complexity. It will be useful in the following to define a Boolean function which indicates whether two spherical polygons intersect (or overlap). We call this function o and define it such that $o(A_{i,j},A_{k,l}) =$ "true" if and only if the two polygons have a point in common.

Table 1. Performance of the Sieve Algorithm Illustrated on Five Real Proteins[a]

protein	N	no. of triplets	time/s	memory
1a5j01	55	729 040	54	1.3M
1cd1C2	94	4 689 011	305	1.7M
14gsA2	107	29 040 558	1 289	1.9M
13pkD2	207	418 225 440	17 231	3.3M
1pamA1	399	7 367 298 819	324 253	8.7M

[a] In every case, $n = 25$, corresponding to 3154 triangles.

The most obvious algorithm for computing the areas we need would be something like

```
 1: for i from 1 to N
 2:  for j from i+2 to N
 3:   for k from i to N
 4:    for l from k+2 to N
 5:     if k*N+l > i*N+j then
 6:      if o(A(i,j), A(k,l)) then
 7:        T(i,j,k,l) = intersection of
                        A(i,j) with A(k,l)
 8:        for m from k to N
 9:         for n from m+2 to N
10:          if m*N+n > k*N+l then
11:           if o(A(i,j),A(m,n)) and
                 o(A(k,l),A(m,n)) then
12:           if o(T(i,j,k,l),A(m,n)) then
13:            compute area(i,j,k,l,m,n)
```

One can of course imagine various possible optimizations, such as first computing and storing the values of o(A(i,j),A(k,l)) for use in lines 6 and 11. What one however very quickly discovers is that such a table would be enormous, with approximately $N^4/2$ entries. For a typical protein length of 150, this corresponds to 250 million entries. While it is true that one can find computers with a large enough memory to handle such a table, the use of such enormous tables inhibits further standard optimizations which could take advantage of actual, physical memory structures (for example, efficient use of the cache would be essentially impossible). Storing only those entries which correspond to true (using techniques similar to those used to store large sparse matrices) would bring this down considerably, but one can still expect such a program to be quite slow due to indirect addressing and still too large for most workstations. The number of calculations involving the overlap function o would be somewhere between $O(N^4)$ and $O(N^6)$, and this gives us some idea of the time complexity.

Must the time complexity of the algorithm we use lie between $O(N^4)$ and $O(N^6)$? Is there a lower bound on the best complexity we can achieve? To answer these questions, we must know how many actual area computations (in line 13 above) would be necessary for a protein of length N. This involves recourse to actual protein data, since geometry alone cannot tell us what geometries real proteins actually have. If we take a few real examples (see Table 1), we find that the number of nonempty intersections (i.e., the number of area computations necessary) grows approximately as $O(N^{4.7})$. Arteca[15] found that for proteins with 52−279 residues the average number of crossings grows as $O(N^{\alpha})$ with $\alpha \approx 1.50 \pm 0.08$, which gives $O(N^{3\alpha})$ as a lower bound on the number

Table 2. Performance of the Sieve Algorithm Illustrated on Five CATH Protein Domains[a]

protein	N	no. of triplets	time/s	memory
1a5j01	55	97 937 280	176	768K
1cd1C2	94	2 858 722 873	3 211	840K
14gsA2	107	5 122 236 685	8 257	776K
13pkD2	207	270 472 231 339	274 128	936K

[a] In every case $n = 1$, corresponding to four triangles and therefore to the naive algorithm introduced in the introduction of the discussion of algorithm development.

of computations necessary. The gap between the central value of $3\alpha = 4.50 \pm 0.24$ and 4.7 is expected, as the number of crossings changes with the point of view. The complexity $O(N^{4.7})$ should be considered to be a lower bound on the time complexity of any algorithm we use. Our aim will be to design an algorithm which has a time complexity very close to this.

As will become clear later, the naive algorithm described above (with some small changes which only lead to greater speed and less memory usage) is actually a special case of the more general algorithm we have developed. We will nonetheless investigate if this naive algorithm has the complexity we desire before describing the more efficient algorithm we have developed.

Table 2 shows the performance of this optimized naive algorithm. We find that its time complexity is approximately $O(N^{5.6})$, which is far worse than we are hoping for, yet not entirely unexpected, as the four outer loops give $O(N^4)$ and the number of actual triples to be found at that point is $\sqrt[3]{O(N^{4.7})}$, giving a complexity of $5.6 \approx 4 + (4.7/3)$.

Memory usage is minimal (see Table 2). This is due to the fact that the algorithm does not create large tables but rather treats candidate triplets at the point where they are generated. Of course it may be that the time complexity could be brought down if one did store a table corresponding to the one discussed above, but it is already clear that this would result in an unacceptable space complexity.

To circumvent this problem of complexity, it is useful to return to the original geometric statement of the problem. We are interested in finding triplets of spherical quadrilaterals which share a common point. The central idea of our algorithm is to use this fact to perform a preselection of triplets, such that a set of candidate triplets is generated, a large proportion of which will actually lead to nonempty intersections. If the generation of this set of candidates can be performed efficiently, then the remainder of the algorithm will have the rather simple task of just eliminating those few candidate triplets which have an empty intersection and computing the areas of the remainder. The hope is that the time complexity of the algorithm should then be dominated by this final phase of computing actual nonzero areas, rather than in computing intersections of quadrilaterals which may never lead to triplets with nonempty intersections. The time complexity should then very closely match the $O(N^{4.7})$ spoken of above.

We perform the preselection spoken of above by using the fact that triplets of nonempty intersection must have a point in common. We cannot deal with the infinite number of points on the surface of the unit sphere, but we can triangulate it with very small triangles (M of them, for instance), and use intersection with these triangles as a preselection criterion. In other words, we can imagine an algorithm which looks like

```
 1: triangulate surface of unit sphere
 2: for p from 1 to M
 3:  clear list L(p)
 4:  for i from 1 to N
 5:   for j from i+2 to N
 6:    if o(triangle(p), A(i,j)) then
        add (i,j) to list L(p)
 7: for p from 1 to M
 8:  for (i,j) in L(p)
 9:   for (k,l) in L(p) so k*N+l > i*N+j
10:    for (m,n) in L(p) so m*N+n > k*N+l
11:     compute area(i,j,k,l,m,n)
```

The immediate advantage of this algorithm is that the complexity of the first phase is only $O(N^2)$, and we of course hope that the time complexity of the second phase will be optimal (i.e., reflecting only the number of nonempty intersections). The only problem we now have is that many triangles may intersect with the intersection of a triplet, and there is a danger of counting triplets' intersections more than once. To avoid this, we take all triangles which intersect a given $A_{i,j}$ and treat them at once, merging the sets of resultant triplets. It is worth mentioning that we have made use of the concept of "lazy lists" (see section 5 of ref 16—although we must point out that we have implemented this by hand in C and not in ML), such that intermediate sets are not actually stored in memory, but their elements are generated in an order which allows them to be used and then discarded immediately. This is the explanation for the excellent space complexity of our algorithm, which is dominated instead by the lists L(p). Concerning the time complexity, we have also structured our algorithm in such a way that if $A_{i,j}$ and $A_{k,l}$ do not overlap, then no attempt is made to construct triplets with them. On the other hand, if they do overlap, then their intersection is stored temporarily and used for all triplets involving them.

It turns out that we only need to triangulate one hemisphere of the surface of the unit sphere. Our program takes a parameter n to define the fineness of the triangularization. The number of triangles M grows essentially as n^2.

Our algorithm turns out to be essentially optimal in time complexity, and almost linearly proportional to the number of triangles in space complexity (see Figure 3). One notices that the number of triangles has an influence on the CPU time (Figure 4). If M is too large, we waste time merging results from the many triangles which cover a nonempty intersection. If M is too small, then our preselection process is too weak, and the final phase of computation has too many candidate triplets to treat. Otherwise, it appears to be a simple matter to choose a reasonable value of M. We find that the time complexity for our algorithm, where we have always chosen the optimal value of M for the given protein, is approximately $O(N^{4.6})$, which is as good as perfect. The fact that it appears to be better than perfect is most probably attributable to a combination of measurement errors and also because our program actually computes first- and second-order measures as well as the third-order measures being discussed here. These first- and second-order measures should be insignificant for long proteins but increase the CPU times for short proteins enough that the algorithm appears to work "too well" on the longer ones.

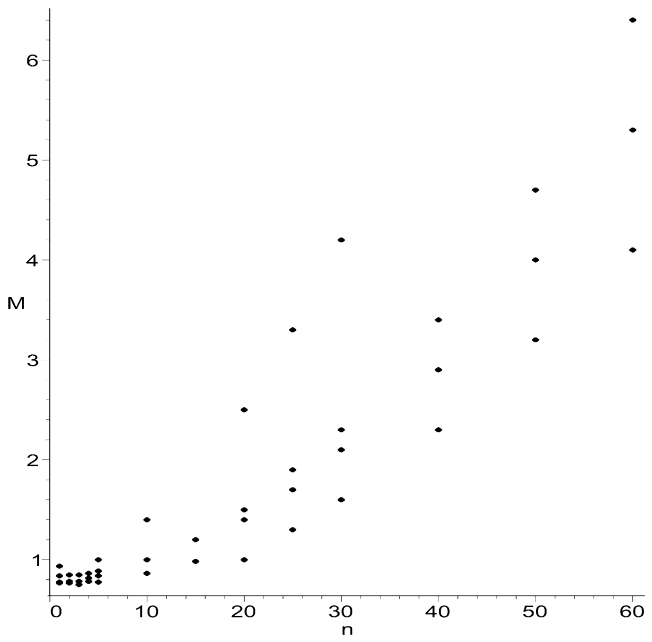

Figure 3. Memory usage (in megabytes) of our algorithm for five different proteins and as a function of *n*. Note that memory usage increases almost (but faster than) linearly for large *n*.

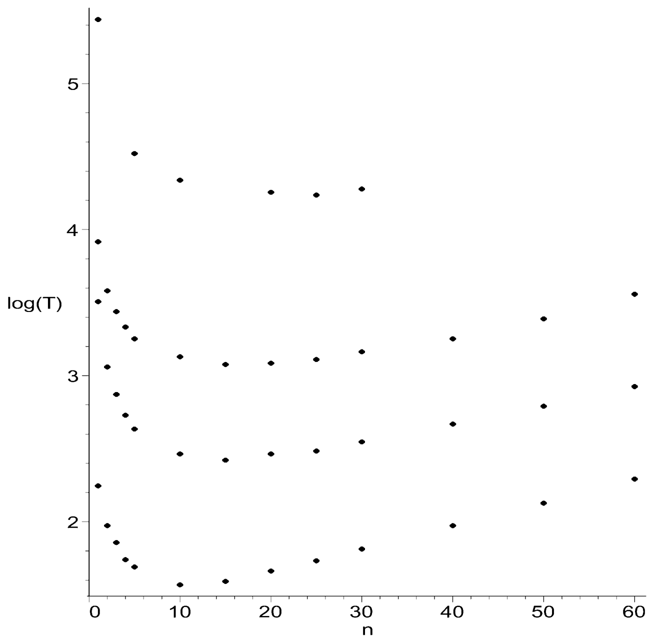

Figure 4. Base 10 logarithm of CPU time (in seconds) of our algorithm for four different proteins of lengths 55 (lower set), 94, 107, and 207 (upper set), as a function of *n*. Note that the shortest execution times are for values of *n* between 10 and 30, growing slowly with the size of the protein.

Numerical Considerations. It is a well-known fact that robust computational algorithms for geometry are difficult to implement in the presence of rounding errors.[17,18] The reason is that geometric statements regarding the intersections of lines and points, etc., cannot be relied upon to be consistent with one another. For example, it may occur that a program claims that two polygons do not intersect, although the same program claims that they have a point in common.

We need a fast algorithm, so rather than implement a robust, general purpose spherical geometry library, we have attempted to combine imperfect algorithms with checks such that the failure of an imperfect calculation is caught automatically, and another algorithm used in its place. This is the reason we will now discuss a number of different approaches to each subproblem. We have also made use of very problem-specific optimizations (we know that certain types of intersection are impossible, for example) which would not be possible in a general purpose library.

We are of course primarily interested in computing areas of spherical polygons with a small number of edges. In the following, we will mainly discuss the case of triangles. See section 18.6 of ref 19 for a survey of known results concerning spherical triangles.

We need a rapid way of determining whether two spherical polygons overlap, and, if so, what their intersection is. For an algorithm which treats the more general problem of finding the intersection of a spherical convex *n*-gon with a spherical convex *m*-gon for large *n* and *m*, see Jong-Sung and Sung-Yong.[20] Since we only consider intersections of sets of convex spherical polygons, we can safely assume that all intersections we compute should also be convex or consist of convex parts. We compute intersections in a number of different phases, ordered such that it is most likely that the routine will respond with “no overlap” as soon as possible, but otherwise continue on to the next phase, until an intersection polygon is computed.

The first phase consists of checking that the signs of the coordinates of the corners of the input polygons are such that an intersection is possible. This is followed by checks to see if corners or edges of each polygon intersect with corners or edges of the other, and which corners of the one polygon are inside the other. If there have been intersections, one can assume that there is a nonempty intersection of the input polygons. In this case, we reuse the information already calculated concerning the intersection of edges and belonging of the one polygon’s corners to the other polygon. However, floating point errors do cause problems here, since one does encounter a situation where these precalculated data are contradictory. For example, if one edge of one polygon is such that its end points are almost exactly on the boundary of the other polygon, it can occur that these end points are deemed to be outside the other polygon, but that the edge does not intersect any edge of the other polygon. From a pure geometrical point of view, this information is misleading, and we were forced to implement a number of heuristics to overcome the problems which arise. These usually involve checking whether the midpoint of the edge in question belongs to the other polygon. Such extra information almost always solves any apparent contradictions.

In those cases where floating point errors leave gaps in the boundary of the computed intersection, one can fill them if they are of a length deemed to be very small (in our case, 10^{-12}). Occasionally, very special cases arise which cannot be solved using the heuristics of our algorithm. These cases can always be identified because they lead to nonsensical boundaries for the computed intersections (for example, edges are missing, or there are isolated edges which cannot be joined to the others). All of these cases turned out to be due to the input data format.

The input data are all given to three decimal places. This means that we can be sure that we are not working in $\mathbb{R}^3$, but in a quite rough, discrete approximation to it. This approximation is due to the finite resolution of the data. The likelihood of the edges of two nonrelated quadrilaterals intersecting at an infinite number of points is fairly high in such a discrete space. Yet in the space of all curves which lie within the absolute errors of the data we have, the likelihood of such an intersection is zero. For this reason, we have decided to deal with exceptional cases by perturbing the input data by an amount which is much smaller than their resolution (we usually perturb by 10^{-10}). In our experience, this removes exceptional cases. Due to the fact that the areas we are computing are continuous functions of the input data, these small changes cannot lead to abrupt changes in the output data. Indeed, we find that the perturbations lead to changes in the computed areas which are not significant, and the relative change of the final output is of order 10^{-10}.

Given the angles α, β, and γ of a spherical triangle, we can compute its area easily by noting that it is equal to its excess $\sigma = \alpha + \beta + \gamma - \pi$. This (and its extension to spherical polygons—the Gauss–Bonnet Theorem) is relatively easy to implement, and we make much use of it in our algorithm, but it suffers from two severe numerical problems. One is that one depends on the computation of determinants to determine the angles involved, and the other is the fact that small—triangles' areas (ϵ) must be computed by what amounts to $(\pi + \epsilon) - \pi$. Both determinants and this type of subtraction lead to severe loss of precision, for example when a triangle is small. In the case of the determinants, the problem is particularly acute when the sign of the determinants involved leads to a miscalculation of one of the angles and hereby the area by some integral multiple of π (an arctangent is involved).

In those cases where our intersection routine delivers a polygon which is not strictly convex (if the boundary contains a small loop due to floating-point errors, for example) the Gauss–Bonnet Theorem method also miscalculates by some integral multiple of 2π.

Such dangerous or difficult cases can however easily be identified, and in such cases we make use of a Heron-type formula for spherical triangles:

$$\mathrm{Tan}\left(\frac{\sigma}{4}\right) = \sqrt{\tan\left(\frac{p}{2}\right)\tan\left(\frac{p-a}{2}\right)\tan\left(\frac{p-b}{2}\right)\tan\left(\frac{p-c}{2}\right)}$$

where a–c are the (arc)lengths of the sides of the spherical triangle and $p = (a + b + c)/2$. The lengths of the sides of a triangle can be computed reliably for small triangles, and there are no determinants involved. In the case of spherical polygons of more than three sides, we can simply divide them up into a number of triangles, for example by using a center of the polygon as a new vertex. This method's main disadvantage is the large number of special function calls. It therefore tends to be slower than computing the excess directly. Otherwise, we would always have used it.

COMPARISON WITH GENERALIZED GAUSS INTEGRALS

The results in this section are based on the average occurrences of crossing patterns and the Gauss integrals calculated for 5965 connected protein domains of CATH version 1.7 that have less than 188 residues, of which at most three are allowed to be missing.

In the case in which a protein looks the same when seen from all directions, one would expect each of the here-considered average crossing pattern occurrences to equal[9] its corresponding Gauss integral. We first test to which extent this average expectation holds for protein structures. Next, we compare the descriptive power of the average occurrences of crossing patterns to that of the Gauss integrals.

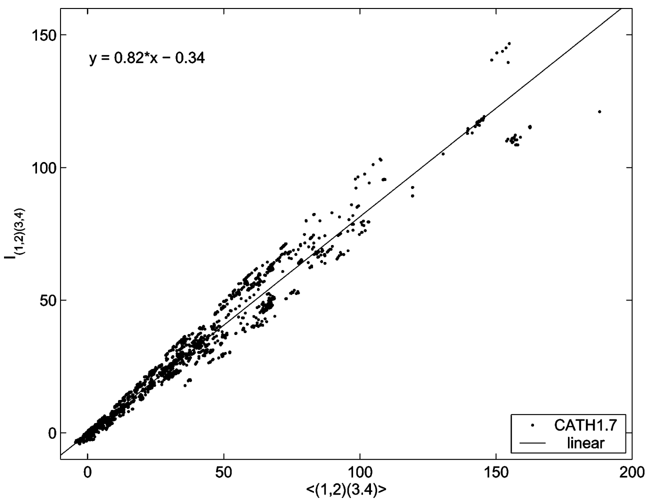

Figure 5. Average occurrence of the (1, 2)(3, 4) crossing pattern versus the values of the Gauss integral $I_{(1,2)(3,4)}$ on 5965 connected protein domains from CATH version 1.7 with at most 188 residues.

Table 3. Correlation Coefficients between Average Crossing Pattern Occurrences and Gauss Integrals

crossing pattern	corr coeff	crossing pattern	corr coeff
(1, 2)	equal	(1, 2)(3, 5)(4, 6)	−0.8838
|1, 2|	equal	(1, 2)(3, 6)(4, 5)	0.3872
(1, 2)(3, 4)	0.9865	(1, 3)(2, 4)(5, 6)	−0.8987
|1, 2|(3, 4)	0.9892	(1, 3)(2, 5)(4, 6)	0.0329
(1, 2)|3, 4|	0.9902	(1, 3)(2, 6)(4, 5)	0.3249
|1, 2||3, 4|	0.9764	(1, 4)(2, 3)(5, 6)	0.5393
(1, 3)(2, 4)	−0.2503	(1, 4)(2, 5)(3, 6)	0.4654
|1, 3|(2, 4)	0.9226	(1, 4)(2, 6)(3, 5)	0.4701
(1, 3)|2, 4|	0.9603	(1, 5)(2, 3)(4, 6)	0.0684
|1, 3||2, 4|	0.9683	(1, 5)(2, 4)(3, 6)	0.3858
(1, 4)(2, 3)	0.9249	(1, 5)(2, 6)(3, 4)	−0.1029
|1, 4|(2, 3)	0.9778	(1, 6)(2, 3)(4, 5)	0.9437
(1, 4)|2, 3|	0.9909	(1, 6)(2, 4)(3, 5)	−0.7744
|1, 4||2, 3|	0.9798	(1, 6)(2, 5)(3, 4)	0.1551
(1, 2)(3, 4)(5, 6)	0.9534		

Correlation. Considering patterns of one crossing, the two average crossing pattern occurrences considered here are exactly equal to the writhe and the average crossing number as when calculated using the Gauss integrals.

The average occurrences of the (1, 2)(3, 4) pattern versus the values of the Gauss integral $I_{(1,2)(3,4)}$ is shown in Figure 5. The two measures are clearly correlated. The three crossing patterns of two crossings each have four distinct sign conventions. Most of the twelve corresponding average occurrences of crossing patterns are clearly correlated with their corresponding Gauss integral. See Table 3. In contrast, the average crossing pattern occurrences involving three crossings are (except for the (1, 2)(3, 4)(5, 6) pattern) not, or at most weakly, correlated with the corresponding Gauss integrals. See Figure 6. However correlation may appear for much longer proteins than those considered here. It is worth mentioning that each Gauss integral and its corresponding average crossing pattern occurrence have the same limit when a backbone is squeezed flat to a planar projection. This common limit is the (integral) number of times that the particular crossing pattern is seen in this projection.

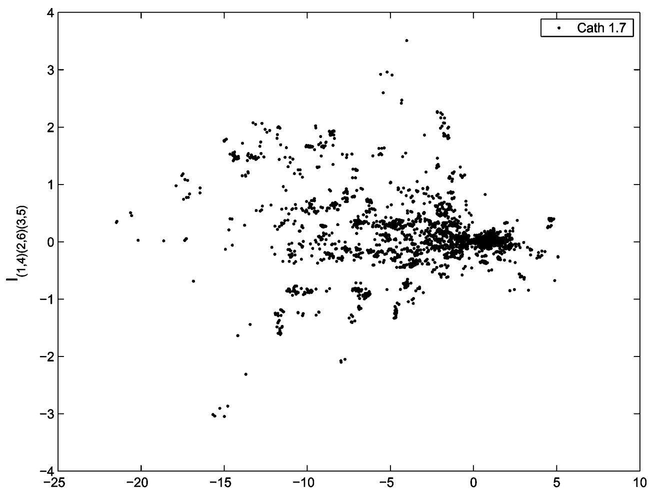

Figure 6. Average occurrence of the (1, 4)(2, 6)(3, 5) crossing pattern versus the values of the Gauss integral $I_{(1,4)(2,6)(3,5)}$ on 5965 connected protein domains from CATH version 1.7 with at most 188 residues.

A surprising observation from Figure 5 is that, for almost all the protein domains, the average occurrences of the (1, 2)(3, 4) pattern are greater than the Gauss integral $I_{(1,2)(3,4)}$. The same holds for the |1, 2|(3, 4), (1, 2)|3, 4|, |1, 2||3, 4|, |1, 4||2, 3|, and (1, 2)(3, 4)(5, 6) patterns, whereas the average occurrences of the (1, 4)(2, 3) pattern are smaller than the Gauss integral $I_{(1,4)(2,3)}$. It is an interesting question if these observations hold for general space curves or if they are peculiar to protein backbones.

Separation of Protein Folds. To treat the measures of average crossing pattern occurrences equally, we normalize each measure to have deviation one on the set of domains from 5965 CATH version 1.7. The usual Euclidean metric on the normalized measures defines a pseudometric on the space of protein structures. We find that the ratio between the average distance between nonhomologous protein domains (low sequence identity) and the average distance between homologous protein domains (high sequence identity) is 4.8, which is comparable with the similar ratio of 5.6 when using the generalized Gauss integrals. Considering protein domains with distinct respectively shared topology, the corresponding ratio is 2.88 when using average crossing configurations and 3.08 when using the generalized Gauss integrals.

By the 96% reproduction of the classification (at the homology level not just at the topology level) of the connected domains of CATH version 2.4 based on the generalized Gauss integrals reported by Røgen and Fain,[10] it follows that each of the two sets of protein structure descriptors captures the diversity of the protein topologies.

FURTHER COMMENTS AND CONCLUSIONS

We have introduced average crossing pattern occurrences as descriptors of protein structures and have developed an asymptotically optimal algorithm for calculating these new descriptors. The main motivation of this work was to examine to which extent each of these new measures correlates with its corresponding Gauss integral. The hope was to find very strong correlations, such that the Gauss integrals would inherit the very intuitive interpretation of the new descriptors. Quite surprisingly, we find that the average occurrences of most patterns with three crossings and even the average occurrences of a pattern with two crossings to be relatively independent of their corresponding Gauss integrals. This independency calls for an involved test comparing the descriptive power of the here-introduced measures and the Gauss integrals, which lies outside the scope of this paper. However, it is probably more interesting to use both the average occurrences of crossing patterns and the Gauss integrals, combined, to improve the description of protein backbones. In the cases of correlated descriptors, it seems especially interesting to consider the difference between the average occurrence of, e.g., the (1, 2)(3, 4) crossing pattern and the corresponding Gauss integral, $I_{(1,2)(3,4)}$, which is an interesting measure of anisotropy of the (1, 2)(3, 4) crossing pattern. An obvious question is why this difference has almost constant sign.

APPENDIX: THE SOFTWARE

Our algorithm has been implemented in ANSI C[21] and uses the standard UNIX system interface model. It is in a single file called Sieve.c, of almost 2000 lines that can be downloaded from http://www.mat.dtu.dk/people/Peter.Roegen/Sieve.html. The program takes five or six arguments. The first two arguments are limits for the length of proteins it should process. For example, 10 20 would indicate that the program should only treat proteins for which $10 < N < 20$. The third argument is the triangulation parameter n. The fourth argument is a directory name (which must end with a "/") in which the program will look for protein data files. The fifth argument is the name of a file into which output data are to be written. The optional sixth argument is the amplitude of random noise to be added to atomic coordinates. If omitted, this amplitude is set to zero. For example, we can use the command

```
> sieve 20 100 25 data/ output 0.000000001
```

The program assumes that protein data files have names ending with ".pdb" and that they are of the pdb-format.[22]

The program searches through the given directory for files ending in .pdb. For each such file, it reads through its output file (which is *not* overwritten, but only appended to) to see if there already is an entry for that protein. If so, it passes over to the next one. If not, it computes the measures for this new protein if it can and appends a line to the output file if it could. The output file is only opened for reading and writing but not during any computation. Once a line is appended to the output file, the output stream is flushed (any buffered but unwritten data are written). This means that the program can be aborted and restarted without losing more than the computation in progress (i.e., one single protein). It also means that one can first set the program to treat a set of proteins without any perturbation of atomic coordinates (i.e., no sixth argument). It will compute the measures of those it can but not produce an output line for those which caused numerical problems. One can then start again with a small perturbation to treat the remainder.

The output consists of a single line per protein, an example being

```
1cd1C2.pdb C 0 95   −2.2006067934   23.21
```

(the line is actually longer). The line begins with the name of the protein data file, followed by the chain name (a "0" will be written here if the chain name is a blank), the number of carbon α atoms missing from the file, and $N + 1$, followed by 29 global measures (first to third order).

The program also writes some information to standard output, such as which files it has looked at, for which of them it could compute the measures, which of them were already in its output file, and how much CPU time the entire computation took.

ACKNOWLEDGMENT

P.R. has been supported by Carlsbergfondet.

REFERENCES AND NOTES

(1) Pohl, W. F. DNA and Differential Geometry. *Math. Int.* **1980**, *3*, 20−27.
(2) Olson, W. K.; Zhurkin, V. B. Modeling DNA Deformations. *Curr. Opin. Struct. Biol.* **2000**, *10*, 286−297.
(3) Sumners, D. W. Untangling DNA. *Math. Int.* **1990**, *12*, 71−80.
(4) Conte, L. L.; Ailey, B.; Hubbard, T. J. P.; Brenner, S. E.; Murzin, A. G.; Chothia, C. SCOP: A Structural Classification of Proteins Database. *Nucleic Acids Res.* **2000**, *28*, 257−259 (http://scop.mrc-lmb.cam.ac.uk/scop/).
(5) Orengo, C. A.; Michie; A. D.; Jones, S.; Jones, D. T.; Swindells, M. B.; Thornton, J. M. CATH−A Hierarchic Classification of Protein Domain Structures. *Structure* **1997**, *5*, 1093−1108 (http://www.biochem.ucl.ac.uk/ bsm/cathnew/index.html).
(6) Getz, G.; Vendruscolo, M.; Sachs, D.; Domany, E. Automated Assignment of SCOP and CATH Protein Structure Classifications from FSSP Scores. *Proteins* **2002**, *46*, 405−415.
(7) Chothia, C. Proteins: One Thousand Families for the Molecular Biologist. *Nature* **1992**, *357*, 543−544.
(8) Govindarajan, S.; Recabarren, R.; Goldstein, R. A. Estimating the Total Nnumber of Protein Folds. *Proteins* **1999**, *35*, 408−414.
(9) Røgen, P.; Bohr, H. A New Family of Global Protein Shape Descriptors. *Math. Biosci.* **2003**, *182*, 167−181.
(10) Røgen, P.; Fain, B. Automatic Classification of Protein Structure by Using Gauss Integrals. *PNAS* **2003**, *100*, 119−124.
(11) Levitt, M. Protein Folding by Restrained Energy Minimization and Molecular Dynamics. *J. Mol. Biol.* **1983**, *170*, 723−764.
(12) Banchoff, T. Self-Linking Numbers of Space Polygons. *Indiana Univ. Math. J.* **1976**, *25* (12), 1171−1188.
(13) Arteca, G. A. Overcrossing Spectra of Protein Backbones−Characterization of 3-Dimensional Molecular Shape and Global Structural Homologies. *Biopolymers* **1993**, *33*, 1829−1841.
(14) Arteca, G. A. Scaling Behaviour of Some Molecular Shape Descriptors of Polymer Chains and Protein Backbones. *Phys. Rev. E* **1994**, *49*, 2417−2428.
(15) Arteca, G. A. Scaling Regimes of Molecular Size and Self-Entanglements in Very Compact Proteins. *Phys. Rev. E* **1995**, *51*, 2600−2610.
(16) Paulson, L. C. *ML for the Working Programmer*; Cambridge University Press: Cambridge, U.K., 1993.
(17) Goldberg, D. What Every Computer Scientist Should Know about Floating-Point Arithmetic. *ACM Comput. Surv.* **1991**, *23*, 5−48.
(18) Yap, C. K. Robust Geometric Computation. In *Handbook of Discrete and Computational Geometry*; Goodman, J. E., O'Rourke, J., Eds.; CRC Press: New York, 1997.
(19) Berger, M. *Geometry II*; Springer-Verlag: Berlin, 1977; Section 18.6.
(20) Jong-Sung, H.; Sung-Yong, S. Edge Advancing Rules for Intersecting Spherical Convex Polygons. *Int. J. Comput. Geom. Appl.* **2002**, *12*, 207−216.
(21) Kernighan, B. W.; Ritchie, D. M. *The C Programming Language*, 2nd ed.: Prentice Hall: Englewood Cliffs, NJ, 1988.
(22) Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description Version 2.1 (draft), October 25, 1996 (http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2frame.html).

CI034095A