EXPLORING PROTEIN FOLDING TRAJECTORIES USING GEOMETRIC SPANNERS

D. RUSSEL and L. GUIBAS
Computer Science Department
353 Serra Mall
Stanford, CA 94305, USA
E-mail: {drussel,guibas}@cs.stanford.edu

We describe the 3-D structure of a protein using geometric spanners — geometric graphs with a sparse set of edges where paths approximate the n^2 inter-atom distances. The edges in the spanner pick out important proximities in the structure, labeling a small number of atom pairs or backbone region pairs as being of primary interest. Such compact multiresolution views of proximities in the protein can be quite valuable, allowing, for example, easy visualization of the conformation over the entire folding trajectory of a protein and segmentation of the trajectory. These visualizations allow one to easily detect formation of secondary and tertiary structures as the protein folds.

1 Introduction

There has been extensive work on visualizing the 3-D structure of proteins in ways that attempt to make the certain aspects of the structure more apparent. For example, commonly used software packages such as `RasMol` [10], `ProteinExplorer` [9], or `SPV` [8], among others, permit visualizations via hard-sphere models, stick models, and ribbon models that emphasize different aspects of the protein surface or secondary structure. Even more abstract visualizations have been used as a tool for understanding intra-molecular proximities, including contact maps and distance matrix images [13]. None of these approaches work very well, however, if the goal is to visualize proteins in motion and not just their static conformations.

Large corpora of molecular trajectories are becoming available through efforts such as *Folding@Home* [12] where molecular simulations are carried out on distributed networks of many thousands of computers. There is an increasing need to compare, classify, summarize, and organize the space of such protein trajectories with an eye toward advancing our understanding of protein folding by studying their ensemble behaviors. Most currently used methods for understanding such data revolve around computing a few summary statistics for each conformation, such as radius of gyration or number of native contacts and watching how these evolve during each trajectory. More similarly, the chemical distance, a statistic of an adjacency graph of the amino acids, was

use to differentiate folded and unfolded states [4]. In this paper we explore the use of a more rich and abstract representation of the protein structure, based on spanners, which makes the task of understanding and exploring the space of protein motions easier.

Our basic idea is to take the continuous folding process and map it to a more discrete combinatorial representation. This representation focuses on higher-level geometric proximities that tend to form and be more stable over time rather than atom coordinates or specific aspects of secondary/tertiary structure. Specifically, we look at the formation of proximities between different parts of the protein across a range of scales, and track the changes of such proximities over time. Our more abstract description of the folding process is in terms of 'proximity events' — when certain proximities are formed or destroyed. Together, these characterize the folding process in a qualitative way and capture the important aspects of the *trajectory*, the sequence of conformations adopted by a protein in a particular folding path. Just as an algebraic topologist captures the essence of the connectivity of a continuous space in a few discrete invariants (the homology groups), we aim to capture the significant conformational changes during motion through a discrete representation of proximities that form and break.

We use *geometric spanners* to accomplish this goal. Starting from an abstract graph with weights on its edges, a spanner is a sparse subgraph (in the sense of having a number of edges roughly proportional to the number of vertices), such that all edges in the full graph can be well approximated by paths in the spanner (in the sense that the sum of the weights of edges of the path in the spanner is very close to the weight of the original graph edge). In the geometric setting the vertices in the original graph are points each pair of which is connected by an edge with weight equal to the Euclidean distance between the corresponding pair of points. The quality of the approximation can be controlled by varying the number of edges in the spanner.

Note that spanners are at once generalizations of contact maps as well as compressions of distance matrices. One can think of a spanner as a multiresolution contact map that allows an approximate reconstruction of the full distance matrix (and therefore the full 3-D structure as well).

We propose to use these combinatorial structures as a tool for capturing the important proximities of a protein conformation and, in this paper, for comparing and visualizing sequences of protein conformations from molecular trajectories. Key properties of the spanner that facilitate these goals include:

- Spanners are proximity based — this parallels proteins where local interactions determine the behavior.

- Spanners are discrete — they have a combinatorial structure whose description does not include any geometric coordinates.
- Spanners are controllable — we can produce descriptors that more loosely or more tightly capture the shape of the protein, converging to distance matrices as the approximation gets tighter.
- Spanners are uniform — there is only one type of combinatorial element, namely an edge. This makes comparison, processing and display simpler.
- Spanners can be made smooth — small changes in the protein conformation generally result in few large changes in the spanner, enabling tracking of the spanner structure over time.
- Spanners are local — the combinatorial features, edges, are affected by a small subset of the total point set. This means that changes in one part of the protein do not generally affect the spanner edges in other parts. As a result the edges can be assigned semantic meaning based on their endpoints, rather than on larger regions of the protein.

We use our spanners to investigate the folding of the protein BBA5 [11] using simulation data produce produced by the *Folding@Home* project. The spanners enable us to produce diagrams which show the formation (and sometimes dissolution) of secondary and tertiary structure during a whole folding trajectory and allow us to segment these trajectories into logical parts. We expect that the spanner approach will provide a valuable toolkit for the understanding and visualization of protein trajectories.

In the next sections we describe how we construct and smooth our spanner-based representation and how we use it to visualize trajectories. Then we discuss our how we have used spanners to try to understand the folding of BBA5. Finally we mention other promising applications of our spanner based representations.

2 Representing Proteins Using Spanners

We first provide a more rigorous definition of a geometric spanner. Let P be a set of points in $\mathbb{R}^3$, Euclidean three-space, and G be a Euclidean graph on P (graph whose vertices are points from P and whose edge weights are Euclidean distances between the endpoints of the edge). For a parameter $s > 1$, known as the *stretch factor*, G is a spanner for P, if for all pairs of points i and j in P with Euclidean coordinates p_i and p_j, $\pi_G(i,j) \leq s||p_i p_j||$ where $\pi_G(i,j)$ denotes the shortest path distance between i and j in the graph G. Thus,

the spanner represents the quadratic number of interpoint distances in P by the much sparser set of edges in G. There is a vast literature on spanners that we will not attempt to review in here any detail; many different spanner constructions are possible. It has been shown that for s arbitrarily close to 1, there exist spanners whose number of edges is proportional to the size of P. The reader is referred to a number of survey papers for background material and additional references [1, 6].

For simplicity, we only use the backbone atoms of the protein. This allows us to meaningfully identify each atom by its index along the backbone, so an edge i, j connects the ith and jth atoms in the backbone. We can identify the edge i, j with the point (i, j) where $i < j$. We define the distance, between two edges i_0, j_0 and i_1, j_1 as the $\mathcal{L}_1$ distance between the points (i_0, j_0) and (i_1, j_1), namely $|i_0 - i_1| + |j_0 - j_1|$. We will write it $d(i_0, j_0, i_1, j_1)$. Two edges are close if they have a small $\mathcal{L}_1$ distance between them the corresponding points. The length as opposed to weight of an edge $l(i, j)$ is defined as $j - i$. Throughout the section s will designate the stretch factor. A s-spanner is a spanner with stretch factor s.

2.1 Computation

We use what is known in the literature as the 'greedy' spanner. Its computation is conceptually very simple: starting with graph G initially containing only the points P, test each of the $\binom{|P|}{2}$ interpoint *candidate edges* for inclusion, ordered from shortest to longest. For each candidate edge i, j, check if $s\|p_i p_j\| < \pi_G(i, j)$. If so, add the edge i, j to the G. We call this test the *inclusion test*. The algorithm runs in $O(n^3)$ time due to the quadratic number of edges and the worst case linear time required to evaluate the inclusion test.

This greedy spanner construction has been shown to have asymptotically optimal complexity (number of edges) and weight (the sum of the lengths of the edges) as well as good practical complexity and weight [2]. Having low weight is important in our context since we want the spanner to consist of as many short edges as possible in order to capture local interactions. Euclidean spanners can be also be produced in $O(n \log^2 n)$ time with the same asymptotic edge count and weight bounds [3] although we have not implemented such methods.

If implemented naively, performing the inclusion test for long edges dominates the running time as it requires a nearly linear time graph search for each of these edges. However, such long edges are extremely unlikely to be in a spanner of a packed protein. If we maintain an upper bound on the graph distance, $d_G(i, j) \geq \pi_G(i, j)$ between each pair of points, i, j, then we can quickly eliminate any candidate edge for which $s\|ij\| > d_G(i, j)$. We can similarly

prune many of the paths while searching.

These upper bounds can be maintained lazily as graph searches are performed. To tighten the upper bounds and further accelerate the process, it is advantageous to periodically compute the all atoms shortest path distances in the current spanner (an $O(n^2)$ process). In addition, we guide the search using the Euclidean distance as a lower bound on the graph distance to bias our search direction, as in the graph search algorithm $A*$. Using these heuristics, the 2-spanner of an 800 atom backbone of a protein can be computed in about a second.

The kinetic spanner proposed in [7] is a possible alternative. It can be cheaply maintained as the underlying points move around. However, it is non-canonical, making comparison between trajectories tricky and it has more long edges, which are hard to assign biological meaning.

2.2 Spanners of Proteins

Figure 1 shows spanners computed using different expansion factors for single protein and gives an estimate of the number of spanner edges per point for typical proteins in their native state.

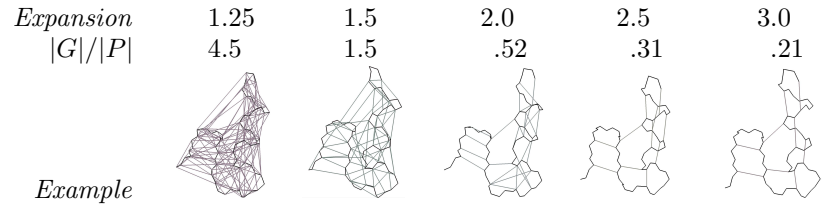
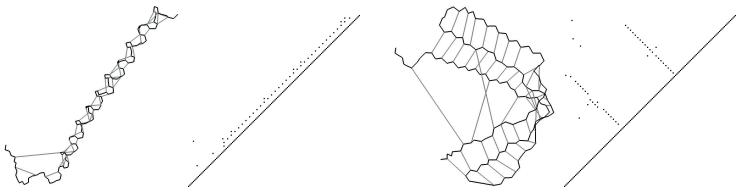


Figure 1: Example spanners and average edge per point for various expansion factors. We mostly use 2-3 spanners for out computations and visualizations as spanners below 2 get very dense.

Secondary structure creates very well defined patterns in the spanner. If each edge is visualized as a point (i, j), then α helices appear as a sequence of points $(i + kq, j + kq)$ where k is a counter variable and q is a stepsize which depends on the expansion factor. For a expansion factors between 2 and 3 the step size is 3, the edges are just longer when the expansion factor is larger. For a expansion factor of 1.5, the stepsize is still 3 but there are several edges leaving from each of the points. β hairpins appear as series of points heading in an orthogonal direction to helices, namely, $(i+kq, j-kq)$. k is 2 for 2 spanners and rises to 4 or 6 for 3 spanners (depending on how the hairpin twists). Both patterns are shown in Figure 2.



(a) Mostly α, 1AQ5 (b) Mostly β, 1YTFD2

Figure 2: Helices and hairpins both give rise to a distinctive pattern of spanner edges: The 2-spanners and corresponding point patterns are shown for two proteins. The backbone edges $i, i+1$ are displayed in addition to the other spanner edges.

3 Filtering Noise

Some addition processing must be done before we can use the spanner for detecting the creation and destruction of long-lasting structural patterns and proximities. Simply computing the spanner for each frame independently results in a sequence of different structures for each frame as the atoms vibrate. We need to be able to distinguish between edges which 'move' by a small amount between frames and edges which are entirely new at a given frame. Once we have done that we can filter the edges based on whether they represent long-lasting proximities.

3.1 Matching Spanners

We can set up the problem of associating edges from one frame with edges from the next as a bipartite matching problem. Such problems can be solved efficiently for a wide variety of similarity metrics. The similarity metric we used with the most success is to make the score for the edges e_0 and e_1 be $\min(l(e_0), l(e_1))/(1 + d(e_0, e_1))$. This allows longer edges to move more easily than shorter ones. In order to avoid spurious matches, we disallow matches of edges which are far from one another, specifically edges which are more than $\min(l(e_0), l(e_1))/3 + 2$ apart. This threshold allows helix and strand edges to be readily matched with one another over the expansion factors we use, but cuts off long distance matches.

Using this technique we can associate edges from successive pairs of frames. We call a sequence of paired edges a *proximity*. The *lifetime* of a proximity is how many different consecutive frames contain edges that contributed to

that proximity. We can now talk about the creation and destruction of a proximity—exactly the type of proximity events we mentioned in the introduction.

3.2 Filtering Noise

Many of the proximities tracked using the previously mentioned technique will have very short lifetimes, perhaps only one frame. These short-lived proximities form a sort of topological noise, which can obscure the real signal in the data.

In order to remove this noise we turn to the idea of persistence [5]. A proximity is *persistent* if its lifetime exceeds a certain threshold. Persistent proximities represent stable aspects of the conformation of the protein backbone. We simply drop non-persistent proximities.

There are certain cases where persistence filtering removes too many edges. For example, if the protein conformation is such that two candidate edges, both of which independently pass the inclusion test, have nearly the same length, then small perturbations can cause either edge to be included in the spanner. The spanner can rapidly alternate between the two edges in succeeding frames, resulting in neither of them being persistent and both of the edges being dropped. Dropping them both looses too much information about the protein conformation. To solve this, in addition to matching edges from the current frame against the edges from the previous frame, we add in an edge from each proximity that was destroyed in a recent frame. As a result, in the previously mentioned example of two edges flipping back and forth, both will now be included.

In practice, we discard proximities which exist for fewer than 20 frames and allow edges to be matched against edges which disappeared up to 6 frames before.

3.3 Segmenting Trajectories

We can now define two spanners as being close if they have similar persistent proximities. This gives us a means of segmenting trajectories into logical components. To do this, assign each consecutive pair of trajectories a *spanner distance*—the sum of the lengths of the persistent proximities which are created or destroyed between this pair of trajectories. We can then smooth these distances in time and use local maxima to segment the trajectory. We will discuss the segmentations produced by this method more in the next sections once we present our technique for displaying spanners.

4 Encoding Spanners in One Dimension

Simply displaying the spanner along with the backbone helps make interactions easier to pick out, especially in less ordered states. However, by itself it does make it easy to visualize the whole trajectory since each frame must be displayed successively. To avoid that problem we want to encode the spanner as a one dimensional object so that we can lay out a whole series of them for inspection together.

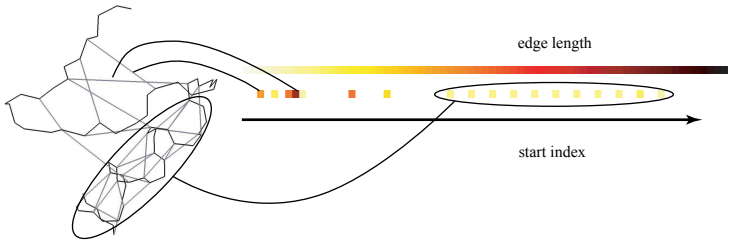


Figure 3: Encoding a spanner into a strip spanner: a 2-spanner and corresponding strip spanner are drawn and some edge and structure correspondences are indicated.

To encode a spanner for a frame, take each spanner edge which is part of a persistent proximity and mark the first point of the edge with the length—i.e. for edge $i, j : i < j$ set $\mathcal{S}[i] = l(i, j)$. We call the resulting vector, $\mathcal{S}$ a *strip spanner*. An example is shown in Figure 3.

One drawback of the strip spanner encoding is that it cannot directly handle more than one spanner edge originating a single point. This can be remedied in one of several manners. One solution is to expand each backbone atom into a constant number of points, i.e. atom i goes to $i - \epsilon, i, i + \epsilon$. This allows a constant number of edges per atom. Alternatively, spanner construction can enforce the one edge per point restriction. For expansion factors 2 and above, this does not significantly distort the spanner, however, it causes problems for factors much below 2 as there are too many edges. For the purposes of generating strip spanners for this paper we simply take the maximum length edge originating at each point. For a small protein such as BBA5 where there are few tertiary edges, this is quite adequate.

Strip spanners from successive frames can then be stacked creating a view of the trajectory as a whole. We call this stacking a *strip history*. An example is shown in Figure 4.

The strip history gives us a good way of displaying the trajectories which were segmented using the technique presented in Section 3.3. An example segmentation is shown in Figure 5. The results of segmentation corresponds

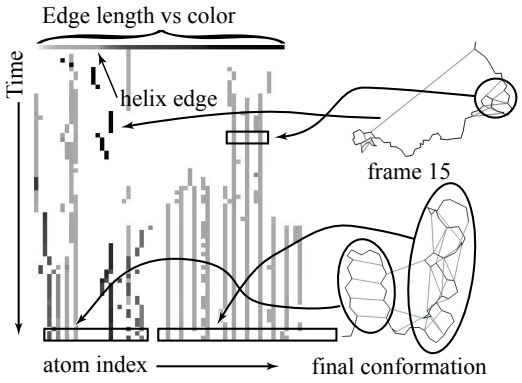


Figure 4: A strip history: In this trajectory, the protein does not fold completely, but forms significant secondary and tertiary structure. The patterns for α helical secondary structure are quite easily seen (lines 3 units apart) on the right half of the diagram. The β hairpin pattern is slightly less easily picked out on the left (lines 2 units apart). Long tertiary edges show up as the darker pixels on the left half of the diagram.

quite well with manual segmentations we had previously performed using the strip history.

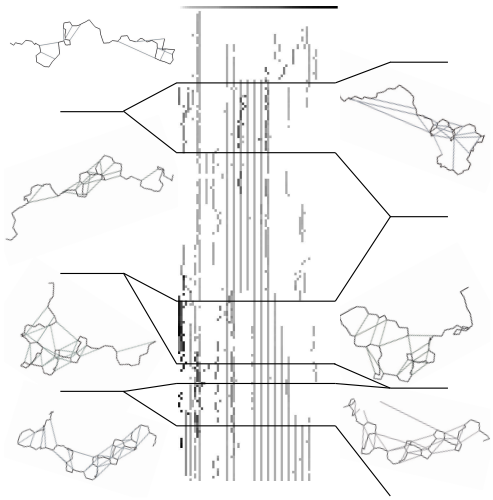


Figure 5: Subdivision of a trajectory using the strip history: A non-folding trajectory for BBA5 was divided into 7 phases using the strip history. The conformation for the middle frame of each phase is shown, always oriented with the α-helical end to the right. During the first segment the protein has little to no secondary structure. In the second the first part of the alpha helix forms and stays fairly static through the third. In the fourth segment we see the hairpin forming, but part of the helix disintegrates. The hairpin disintegrates in the fifth. In the seventh the hairpin reforms, and the end of the helix forms for the first time.

5 Understanding BBA5 Trajectories

We applied our spanner descriptor to the task of trying to understand the folding process of the protein BBA5. It is a 23 residue protein which folds comparatively quickly. The native state of BBA5 consists of a β hairpin, involving the first 8 residues and an α helix involving the last 10 residues.

The hairpin is packed against the helix, although not very tightly due to the presence of large sidechains between the two pieces of secondary structure.

Our data consists of 23 trajectories with frames every 200fs. Thirteen of the trajectories come within 3.1Å cRMSD$_{C_\alpha}$ of the native structure.

Figures 4 and 5 show strip histories for two example trajectories. The times of formation of the α helix are quite clearly visible. The β hairpin formation is less clearly defined, but still can be seen on the left of the image, as can the occasional large backbone distance tertiary interactions, which are the points colored the darkest.

Perusal of the strip histories show that there are few constants among the folding trajectories, as might be expected given its small size.

- α helical secondary structure rarely went away once created, although there were a few cases where parts of the helix formed and then dissolved (such an example can be seen in Figure 5). This may be an artifact of how trajectories were selected for storage. Parts of the helix formed in all possible orders.
- β hairpin structure was much less stable than the helix, however it did stabilize without the presence of tertiary interactions in several of the trajectories.
- tertiary interactions formed before secondary structure in some but after strong secondary structure in others.

In one of the strip histories a helix like pattern can be seen forming in the loop end of the hairpin. It persists for 30 frames. Direct inspection of the backbone structure confirmed that there is indeed a one and a half turn helix like structure there. This trajectory is not shown here. We did not observe similar structure formation in any of the other trajectories.

6 Future Work

Sidechains play an important role in the protein folding process. For example, in BBA5, the rings from a phenylalanine and a tyrosine occupy much of the space between the helix and the hairpin in the native state, making the helix and hairpin pack much less tightly than they would in the absence of such rings. The position of the sidechains is currently ignored in our calculations, although it may be important to the folding process and ignoring it makes the tertiary structure show up much less clearly. Simply adding the whole sidechains adds too many new edges to the spanner making the relevant data hard to extract and disrupts the linear order which we depend on for matching

and visualization. A better approach may be to add a single point per sidechain which will capture its location without complicating the structure too much.

The strip spanner is a one dimensional descriptor which captures key aspects of the proteins conformation. Searching and matching of one dimensional structures is a much easier problem than matching three dimensional curves, suggesting that the strip spanner might have applications in protein structure motif searching and structure alignment. However, there are a number of issues with incorporating gaps which need to be resolved.

We are trying to apply spanners to the problem of understanding the parts of protein conformation space relevant to folding. The strip history based segmentation provides one way of dividing trajectories into chunks which could be matched against one another to find common paths through conformation space. There are a number of problems with measuring the distances between spanners which need to be resolved first. In addition, we suspect simple proteins such as BBA5 fold too quickly and have too small an energy barrier for its fold space to have significant structure. As a result we plan to apply the techniques to unfolding data.

Acknowledgments

This work has been supported by NSF grants CARGO 0310661, CCR-0204486, ITR-0086013, ITR-0205671, ARO grant DAAD19-03-1-0331, as well as by the Bio-X consortium at Stanford.

The authors would like to thank Rachel Kolodny for many valuable suggestions regarding the project and Vijay Pande for providing the data.

References

[1] S. Arya, G. Das, D. Mount, J. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 489–498. ACM Press, 1995.

[2] G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Symposium on Computational Geometry*, pages 53–62. ACM Press, 1993.

[3] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. In *Symposium on Computational geometry*, pages 132–139. ACM Press, 1994.

[4] Nikolay Dokholyan, Lewyn Li, Feng Ding, and Eugene Shakhnovich. Topolical determinants of protein folding. *Proceedings of the National Academy of Science*, pages 8637–8641, 2002.

[5] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Compututational Geometry*, 28:511–533, 2002.

[6] D. Eppstein. Spanning trees and spanners. *Handbook of Computational Geometry*, pages 451–461, 2000.

[7] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. In *Symposium on Computational Geometry*, pages 179–199, June 2004.

[8] N. Guex and M. C. Peitsch. Swiss-model and the swiss-pdbviewer: An environment for comparative protein modeling. *Electrophoresis*, 18:2714–2723, 1997. URL `http://www.expasy.org/spdbv/`.

[9] E. Martz. Protein explorer: Easy yet powerful macromolecular visualization. *Trends in Biochemical Sciences*, 27:107–109, 2002. URL `http://proteinexplorer.org`.

[10] Rasmol. URL `http://www.umass.edu/microbio/rasmol/`.

[11] Y.M. Rhee, E. J. Sorin, G. Jayachandran, E. Lindahl, and V. Pande. Simulations of the role of water in the protein-folding mechanism. In *Proceedings of the National Academy of Science*, volume 101, pages 6456–6461, 2004.

[12] M. Shirts and V. Pande. Screen savers of the world, unite! *Science*, 2000. URL `http://folding.stanford.edu/`.

[13] M. J. Sippl. On the problem of comparing protein structures. *Journal Molecular Biology*, 156:359–388, 1982.