

# Combining Discrete and Continuous Optimization to Detect Protein Substructure Similarity (two page abstract)

L. Paul Chew<sup>1</sup>

A protein can be considered as a string (on the alphabet of 20 amino acids) or as a structure (each protein folds into a particular 3D configuration). A protein's function is largely determined by its shape; thus, there is wide interest in techniques for analyzing protein structure. Ideally, one should be able to analyze proteins as structures as easily as one can analyze proteins as strings.

Consider the following string-based problem: Given two protein strings that are not necessarily similar in their entirety, determine the *most similar* contiguous substrings, one from each protein. In [Gusfield 1997] this is called the *local alignment* or *local similarity* problem. The exact meaning of *most similar* here is determined by the user; it is based on user-specified scores (1) for character vs. character similarity and (2) for character vs. space similarity. It is important to allow for spaces or gaps because evolutionary changes to proteins often involve insertion or deletion of one or more individual amino acids.

The goal here is to design an algorithm for local similarity of protein *structures* as opposed to protein strings. Our algorithm design inspiration is drawn from the DP-based local similarity algorithm for strings. Instead of comparing sequences of characters, we compare sequences of vectors. One complication for working with structures instead of strings is the problem of orientation: basically, two structures or substructures that have similar shape can “look different” if they are at different orientations. Algorithmically, this means that we must establish the optimal orientations for our two proteins as well as finding the similar subsequences. In other words, an algorithm for local similarity of structures involves both discrete optimization (to find the corresponding subsequences) and continuous optimization (to find the optimal orientation). Interestingly, if the correspondence is given then the optimal orientation (for that correspondence) is easy to find, and if the the orientation is given then the optimal correspondence (for that orientation) is easy to find. The challenge is to accomplish both optimizations at once. Note that the technique presented here produces a globally optimal solution; there are no approximations or assumptions of randomness

**Dynamic Programming.** The variant of Dynamic Programming (DP) that we use here finds the best substring match between two sequences based on a *similarity table*. Given two input strings, the goal is to extract the most similar pair of substrings, one from each input string. Such substrings can exist even though the input strings in their entirety are not particularly similar. Finding the highest scoring substrings is equivalent to finding the highest scoring path within the DP similarity table. For sequences of length  $m$  and  $n$ , the similarity table is an  $m$  by  $n$  table in which position  $(i, j)$  holds  $s(i, j)$ , the similarity score for aligning the item at position  $i$  in the first sequence to the item at position  $j$  in the second sequence. A path in a DP similarity table consists of diagonal, horizontal, and vertical segments connecting *nodes*. Intuitively, a node is a junction between the horizontal and vertical lines that are used to draw the table. DP is used to find the highest-scoring path in this kind of table in  $O(mn)$  time where  $m$  and  $n$  are the lengths of the two sequences being compared.

---

<sup>1</sup>Computer Science Dept, Upson Hall, Cornell University, Ithaca, NY 14853 [chew@cs.cornell.edu](mailto:chew@cs.cornell.edu)

**Dynamic Programming and Unit Vectors.** Our goal is to compare two proteins by comparing their sequences of unit vectors. The compared vectors are those between adjacent  $\alpha$ -carbons along the protein backbone. We refer to these direction vectors as *unit* vectors because, for proteins, these vectors are all the same length (about  $3.8\text{\AA}$ ). By chaining the unit vectors head-to-tail, we obtain the protein's backbone as a sequence of  $\alpha$ -carbons in space. Alternatively, we can place all of the unit vectors at the origin; the protein backbone is thus mapped into vectors in the unit sphere. The strategy is to use DP for the comparison, modeling our ideas after the way in which DP is used to find the best substring match for two strings. We require that similar vectors produce a high similarity score; this is necessary for the kind of local similarity matching that is our goal. Thus, we define the similarity score for  $u$  aligned with  $v$  to be  $u \cdot v$ .

This immediately leads to a way to use DP to find the optimal correspondence between two proteins *with fixed orientations*. We can produce a dot-product-based similarity matrix for the two sequences of unit vectors, choose a reasonable gap penalty, and then run the DP algorithm. The problem is that the proteins are not necessarily presented at the optimal orientation. Of course, an optimal orientation can be found by using the RMS algorithm, but this requires that we first know the correspondence between unit vectors. In other words, we can find an optimal orientation if we first know the correspondence and we can find an optimal correspondence if we first know the orientation. The goal of this paper is to accomplish both optimizations at once.

One potential solution is to iterate, first finding a (crudely accurate) correspondence, using this to find an orientation which is then used to improve the correspondence, etc. This works within a single protein *family* because all the members of the family have similar shape; thus, relatively crude methods can be used to establish initial orientations. In general, the iterative method can become caught in local minima.

**Results.** For a 2D analog of the 3D protein-matching problem, DP for unit-vectors can be viewed as operating on complex numbers. This view makes it possible to “isolate” the effects of rotation. By isolating the rotation we can treat an entry on the DP path as a set of points in 3D (as opposed to standard DP for strings where an entry on the DP path is a single number indicating the score for the path-so-far). Three dimension are needed because we need to represent both the complex plane and an extra dimension used to indicate the accumulated gap-penalty. A diagonal path-segment corresponds to a shift of the points in the complex plane; a horizontal or vertical path-segment corresponds to a shift perpendicular to the complex plane. We show that the only points that can lead to an optimal solution are those that are extreme points (i.e., they are points on the convex hull); thus, we can reduce the number of points that must be maintained along the DP path.

For the 3D version of the problem, the version that corresponds to real proteins, we use matrices in place of complex numbers. Ideally, one would prefer to use quaternions since 3D-rotation can be easily represented using quaternion multiplication, but quaternions require two-sided multiplication to represent a rotation and this two-sided multiplication makes it impossible to isolate the rotational term. Using matrices, DP for 3D unit-vectors can be done using points in 10 dimensions (9 dimensions are used to represent a 3-by-3 matrix and the extra dimension is used to encode the gap-penalty). The necessary extreme-point algorithm runs in time  $O(n^2)$  where  $n$  is the number of points (this is based, though, on the linear-time algorithm for Linear Programming which has a large constant factor that depends on the dimension).

The 2D algorithm has been implemented in Matlab. The 3D version has not yet been implemented.